

**Oracle HTML DB Handbook (ISBN: 0-07-225768-7)****Web Appendix****PL/SQL Web Toolkit and Packages****Structure Tags**

At the base of the PL/SQL Web Toolkit are the Structure tags. For example:

```
create or replace procedure structure_example is
begin
  http.htmlOpen;
  http.headOpen;
  http.title('Structure Example');
  http.base('http://laptop2/');
  http.meta('Expires',NULL,'Fri, 18 Jul 1997 04:15:15 MST');
  http.meta('Keywords',NULL,'TUSC, Ultimate, Oracle, Consulting');
  http.meta('Reply-to',NULL,'brownb@tusc.com (Bradley D. Brown)');
  http.headClose;
  http.bodyOpen('/bg_image.gif');
  http.header(1,'Structure Example');
  http.print('This is a structure example, notice the background image too.');
```

***http.htmlOpen***

Prints a tag that indicates the beginning of an HTML document. The beginning HTML tag (<HTML>) starts an HTML file, and the ending HTML tag (</HTML>) ends the HTML file. These tags are used to enclose the entire HTML document.

**Syntax**

```
http.htmlOpen;
```

**Attributes**

VERSION, URN, ROLE

**Generates**

```
<HTML>
```

***http.htmlClose***

Prints a tag that indicates the end of an HTML document.

**Syntax**

```
http.htmlClose;
```

**Generates**

```
</HTML>
```

***http.headOpen***

Prints a tag that indicates the beginning of the HTML document head. This tag specifies that the lines within the beginning (<HEAD>) and ending (</HEAD>) tags are the prologue to the rest of the file. In other words, the HEAD contains information about the HTML document itself. The document head typically contains few tags. The most common tags found in the document head are the <TITLE> and <META> tags.

Tip: Never put body text in the document head.

Tip: JavaScript functions, variables, and document-write commands normally go in the HEAD section.

The HEAD element has no attributes, and the start and end tags according to the HTML 3.0 specs can always be safely omitted as they can be readily inferred by the parser. However, for compatibility it is recommended that you use them. Information in the HEAD element corresponds to the top part of a memo or mail message. It describes properties of the document such as the title, the document toolbar, and additional meta information. There is no intended significance to the order of elements in the document head.

The TITLE element is highly recommended within the HEAD section. If you don't include a TITLE statement in your HEAD section, most browsers use the URL as

the TITLE. If the user saves a bookmark, the URL is saved as the name too. Only certain elements are allowed within the HEAD element.

**Syntax**

```
http.headOpen;
```

**Generates**

```
<HEAD>
```

***http.headClose***

Prints a tag that indicates the end of the HTML document head.

**Syntax**

```
http.headClose;
```

**Generates**

```
</HEAD>
```

***http.bodyOpen***

Prints the tag that identifies the beginning of the body of an HTML document. It allows you to specify an image in the background of the document. This tag encloses the body (i.e., the text and tags) of the HTML document. Note that if you use JavaScript, your Onload and OnUnload functions (for BODY) should be defined in the HEAD section.

**Syntax**

```
http.bodyOpen (cbackground, cattributes);
```

**Attributes**

ID, LANG, NOWRAP, CLEAR, LANG, CLASS, BGCOLOR, TEXT, LINK, ALINK, VLINK

**Generates**

```
<BODY background="cbackground" cattributes>
```

## ***http.bodyClose***

Prints a tag that indicates the end of the HTML document body.

### **Syntax**

```
| http.bodyClose;
```

### **Generates**

```
| </BODY>
```

## ***http.Comment***

Prints an HTML tag that allows you to store comments or lines in HTML pages. These comments are not visible to the end user.

To include comments in an HTML document that will be ignored by the parser, surround them with `<!--` and `-->`. After the comment delimiter, all text up to the next occurrence of `-->` is ignored. This is why comments cannot be nested. Whitespace is allowed between the closing `--` and `>`, but not between the opening `<!` and `--`. For example:

```
| <HEAD>  
| <TITLE>Introduction to Oracle Web Server</TITLE>  
| <!-- Id: Text.html,v 1.6 1997/10/25 23:33:48 brownb Exp -->  
| </HEAD>
```

Some historical implementations incorrectly interpret a `>` sign as a terminate comment.

### **Syntax**

```
| http.Comment (ctext);
```

### **Generates**

```
| <!-- ctext -->
```

## Head-related Tags

### *http.base*

Prints an HTML tag that records the full URL of the current document. The CTARGET attribute establishes a default window name to which all links in this document are targeted. The URL of the current document is useful information if you have a menu in one frame that loads documents into another frame.

The BASE element allows the URL of the document itself to be recorded in situations in which the document may be read out of context. URLs within the document may be in a “partial” form relative to this base address. The default base address is the URL used to retrieve the document.

#### Syntax

```
| http.base (ctarget, cattributes);
```

#### Generates

```
| <BASE HREF={current URL} TARGET=ctarget cattributes>
```

### *http.isindex*

Creates a single entry field with prompting text, such as “enter value,” and then sends that value to the URL of the page or program. This tag indicates that this document is a gateway script that allows searches. The ISINDEX element informs the HTML user agent that the document is an index document. The user can read the ISINDEX tags or use them in a keyword search by adding a question mark to the end of the document address.

Oracle WebServer doesn’t generate the ISINDEX tag automatically. If added by hand to an HTML document, the browser assumes that the server can handle a search on the document. Therefore, simply adding <ISINDEX> in the document is not enough to make searches happen if the server does not have a search engine!

#### Syntax

```
| http.isindex (cprompt, curl);
```

**Generates**

```
<ISINDEX PROMPT="cprompt" HREF="curl">
```

***htp.linkRel***

Prints the HTML tag that gives the relationship described by the hypertext link from the anchor to the target. This is used only when the HREF attribute is present. This tag indicates a relationship between documents but does not create a link. To create a link, you must use the `htp.anchor` command. The relationship is only a logical relationship between HTML documents. It is used primarily by tools such as FrontPage to create a hierarchical graphic representation of HTML documents and therefore is primarily used for static HTML.

The LINK element indicates a relationship between the document and some other object. A document may have any number of LINK elements. The LINK element is empty (does not have a closing tag) but takes the same attributes as the ANCHOR element. This tag is generally used by HTML-generation tools such as FrontPage. <LINK> creates links from one document to the complete text of another document. Anchors, on the other hand, create multiple links with the document.

**Syntax**

```
htp.linkRel (crel, curl, ctitle)
```

**Generates**

```
<LINK REL="crel" HREF="curl" TITLE="ctitle">
```

***htp.linkRev***

Gives the relationship described by the hypertext link from the target to the anchor. This is the opposite of `htp.linkRel`. This tag indicates a relationship between documents but does not create a link. To create a link, you must use the `htp.anchor` command. This is the child pointing back to its parent and is primarily used by tools such as FrontPage for managing static HTML files.

The LINK element indicates a relationship between the document and some other object. A document may have any number of LINK elements. The LINK element is empty (does not have a closing tag) but takes the same attributes as the ANCHOR element. This tag is generally used by HTML-generation tools. <LINK> creates links from one document to the complete text of another document. Anchors, on the other hand, create multiple links with the document.

### Syntax

```
http.linkRev (crev, curl, ctitle);
```

### Generates

```
<LINK REV="crev" HREF="curl" TITLE="ctitle">
```

### *http.meta*

Prints an HTML tag that identifies and embeds document meta information (information about the HTML script) that supplies the web browser with information about the objects returned in HTTP.

The <META> tag indicates information or data about the HTML document itself—for example, keywords for search engines, special headers to be used to retrieve documents via HTTP, expiration dates, and so on. Meta information is usually in a pair form with the key and value specified together.

The META element is used within the HEAD element to embed document meta information not defined by other HTML elements. Such information can be extracted by servers and/or clients for use in identifying, indexing, and cataloging specialized document meta information.

In addition, search engine spiders (which are programs that crawl through the Web) can read the contents of the document head corresponding to any elements defining a value for the attribute HTTP-EQUIV. This provides document authors with a mechanism for identifying information that should be included in the response headers of an HTTP request.

If the document contains

```
<META HTTP-EQUIV=Expires CONTENT="Tue, 30 Dec 1997 21:29:02 GMT">
<META HTTP-EQUIV="Keywords" CONTENT="TUSC, Founder, Corporation, Oracle, Inc
500">
<META HTTP-EQUIV="Reply-to" CONTENT="brownb@tusc.com (Bradley D. Brown)">
```

the server will include the following header information:

```
Expires: Tue, 30 Dec 1997 21:29:02 GMT
Keywords: TUSC, Founder, Corporation, Oracle, Inc 500
Reply-to: brownb@tusc.com (Bradley D. Brown)
```

When the HTTP-EQUIV attribute is absent, the server should not generate an HTTP response header for this meta-information:

```
<META NAME="IndexType" CONTENT="Service">
```

### Syntax

```
htp.meta (chttp_equiv, cname, ccontent);
```

### Generates

```
<META HTTP-EQUIV="chttp_equiv" NAME = "cname" CONTENT="ccontent">
```

### *htp.title*

Prints an HTML tag with the text you pass in as the value of TITLE. Most web browsers display the text value enclosed between <TITLE> and </TITLE> at the top of the document-viewing window. Every HTML document should contain a TITLE element. The title should identify the contents of the document in a global context and may be used in a history list and as a label for the window displaying the document. Unlike headings, titles are not ordinarily displayed in the text of a document itself. The TITLE information is used by the most browsers when the user saves a bookmark, too. Therefore, remember that if this TITLE ends up in a browser's bookmark list, it has to make sense to the user at a later date. For example, rather than having a title of "Downloadable Documents", you may want to have something like "TUSC's Downloadable Documents", so that when the user sees this in his bookmark list two weeks from now, it will still make sense to him.

The TITLE element must occur within the head of the document and may not contain anchors, paragraph tags, or highlighting. There may only be one TITLE in any document.

### Syntax

```
http.title (ctitle);
```

### Generates

```
<TITLE>ctitle</TITLE>
```

## Functional Object Tags

The functional object tags are used to indicate links, import graphics and sound, and attach e-mail capabilities.

### *http.anchor*

Prints the HTML tag for an anchor that server as the starting or ending destination of a hypertext link. This anchor can accept several attributes, but either HREF or NAME is required. HREF specifies where to link to. NAME allows the tag to be a target of a hypertext link. I call this an “inner document tag.” It’s similar to a label in a program. In fact, when you use Word 97 to save a Word document as an HTML file, bookmarks are saved as anchor tags. To create bookmarks in Word, you simply move your cursor to the position in the document that you wish to bookmark, select the Bookmark option from the Edit menu, type the name of the bookmark (which ends up as your anchor tag name in HTML), and then click Add (bookmark). This allows you to branch right to a specific area within the Word/HTML document. You use this quite often for context-sensitive help.

The anchor <A> element is used to define the start and/or destination of a hypertext link. In previous versions of HTML, it provided the only means for defining destination anchors within documents. But now you can use any ID attribute as a destination anchor, so links can be made to divisions, paragraphs, and most other elements.

**Syntax**

```
htp.anchor (curl, ctext, cname, cattributes);
```

**Attributes**

```
ID, LANG, CLASS, MD, SHAPE, TITLE, REL, REV, URL, METHODS
```

**Generates**

```
<A HREF="curl" NAME="cname" cattributes>ctext</A>
```

***htp.anchor2***

Prints the HTML tag for an anchor to be the starting or ending destination of a hypertext link. This anchor can accept several attributes, but either HREF or NAME is required. HREF specifies where to link to. NAME allows this tag to be a target of a hypertext link. This procedure differs from htp.anchor in that it provides a TARGET frame to put the URL (or HREF) results into when clicked upon. If you are not using FRAMES, then use htp.anchor.

**Syntax**

```
htp.anchor2 (curl, ctext, cname, ctarget, cattributes);
```

**Attributes**

```
ID, LANG, CLASS, MD, SHAPE, TITLE, REL, REV, URL, METHODS
```

**Generates**

```
<A HREF="curl" NAME="cname" TARGET = "ctarget" cattributes>ctext</A>
```

***htp.area***

Prints an HTML tag to specify the shape of a client-side image map region. To define the actions for the image map (in other words, to define the hyperlink to a hot spot within an image), use either of these two tags:

- <MAP>
- <AREA>

<MAP> defines the hot spots within the image. <AREA> defines the actions for the image map and has four attributes.

**Syntax**

```
http.area(ccoords, cshape, chref, cnohref, ctarget, cattributes);
```

**Generates**

```
<AREA COORDS="ccoords" SHAPE="cshape" HREF="chref" NOHREF TARGET="ctarget"  
cattributes>
```

***http.bgsound***

Prints an HTML tag to include background sound for a web page. This command is valid only for Internet Explorer (IE) 2.0+.

Sound is a great enhancement to your documents. IE and Netscape Navigator both support audio files in Microsoft's WAV format, Sun Microsystem's AU format, or in MIDI format. WAV produces the largest file size and best quality sound. MIDI, which is not really a "sound" file, is similar to a digitized keyboard in that it plays musical notes. In general, MIDI files are small.

The source audio file begins playing when it is fully downloaded. The source audio file is not a streaming audio player. By default, the sound file plays only once. You can change this default if you use the LOOP attribute, which specifies the number of times that the audio file plays. If you wish to insert a delay between each time the file plays, use the LOOPDELAY attribute. For samples of sound options, see <http://www.jinglephone.com>.

**Syntax**

```
http.bgsound(csrc, cloop, cattributes);
```

**Attributes**

```
LOOPDELAY
```

**Generates**

```
<BGSOUND SRC="csrc" LOOP="cloop" cattributes>
```

***htp.img***

Prints an HTML tag that signals the browser to load an image to be placed into the HTML page. ALT allows you to specify alternate text to be shown while the image is being loaded, or to show text instead of the image if the browser does not support images. IE 3.0+ and Navigator 4.0+ also use the ALT text to display when the mouse cursor is over an image. The ISMAP attribute indicates that the image is an image map.

**Syntax**

```
htp.img (curl, calign, calt, cismap, cattributes);
```

**Attributes**

```
ID, LANG, CLASS, MD, WIDTH, HEIGHT, UNITS, VSPACE, HSPACE, BORDER, LOWSRC,  
DYNSRC, CONTROLS, LOOP, LOOPDELAY, START
```

**Generates**

```
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP cattributes>
```

***htp.img2***

Prints an HTML tag that signals the browser to load an image to be placed into the HTML page. ALT allows you to specify alternate text to be shown while the image is being loaded, or to show text instead of the image if the browser does not support images. The ISMAP attribute indicates that the image is an image map. The CUSEMAP parameter specifies a client-side image map. The *htp.img2* procedure supports the USEMAP attribute, whereas the *htp.img* procedure does not.

The <IMG> tag is used to incorporate inline graphics (typically icons or small graphics) into an HTML document. For example:

```
<IMG SRC="tajmahal.gif" ALT="The Taj Mahal">
```

Browsers that cannot display inline images ignore the IMG element unless it contains the ALT attribute. Note that some browsers can display (or print) linked

graphics but not inline graphics. If the graphic is essential, you may want to create a link to it rather than to put it inline. If the graphic is decorative, then using `IMG` is appropriate. This element is *not* intended for embedding other HTML text. For large figures with captions and text flow, see the `FIG` element.

### Syntax

```
http.img2 (curl, calign, calt, cismap, cusemap, cattributes);
```

### Attributes

```
ID, LANG, CLASS, MD, WIDTH, HEIGHT, UNITS, VSPACE, HSPACE, BORDER, LOWSRC,  
DYN SRC, CONTROLS, LOOP, LOOPDELAY, START
```

### Generates

```
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP USEMAP="cusemap"  
cattributes>
```

### *http.mailto*

Prints the HTML tag for an anchor with ‘mailto’ concatenated before the mail address argument. This is another type of anchor tag. It allows you to have an anchor that uses the mailto protocol. Most browsers call MAPI (Mail Application Programmer Interface) that, in turn, brings up an e-mail window with the address filled in.

### Syntax

```
http.mailto (caddress, ctext, cname, cattributes);
```

### Generates

```
<A NAME=cname HREF="mailto:caddress" cattributes>ctext</A>
```

### *http.mapOpen*

Prints an HTML tag to specify a set of regions in a client-side image map.

To define the actions for the image map (in other words, to define the hyperlink to a hot spot within an image), there are two tags:

- `<MAP>`

- `<AREA>`

`<MAP>` defines the hot spots within the image. The only required attribute for the `<MAP>` tag is `NAME`. The `NAME` attribute defines the name of the image map. Use `<AREA>` tags immediately following `<MAP>` tags to indicate the hot spots for the image map.

**Syntax**

```
http.mapOpen(cname, cattributes);
```

**Generates**

```
<MAP NAME="cname" cattributes>
```

***http.mapClose***

Prints an HTML tag that ends the definition of a client-side image map.

**Syntax**

```
http.mapClose;
```

**Generates**

```
</MAP>
```

**List Tags**

List tags allow you to display information in ordered lists with numbered items, in unordered lists that contain bullets to mark each item, and in definition lists, which alternate a term with its definition.

***http.dirlistOpen***

Prints an HTML tag that begins a directory list. This element was superseded by extensions to the `UL` element. User agents are advised to continue to support it for the sake of legacy documents. `DIR` consists of one or more `LI` elements, similar to `UL`. `DIR` lists are used to present lists of items containing up to 20 characters each. Items in this list are arranged in columns, typically 24 characters wide. The `<LI>` or `http.listItem` must appear directly after you use this tag.

You can get the same effect with `<UL PLAIN WRAP=HORIZ>` or `http.ulistOpen(cwrap => 'HORIZ' cattributes => 'PLAIN');`.

**Syntax**

```
http.dirlistOpen;
```

**Generates**

```
<DIR>
```

**Example**

See the previous example.

***http.dirlistClose***

Prints an HTML tag that closes the directory list tag, `http.dirlistOpen`.

**Syntax**

```
http.dirlistClose;
```

**Generates**

```
</DIR>
```

***http.dlistDef***

Prints an HTML tag that is used to insert terms and their corresponding definitions in an indented list format. The `http.dlistTerm` must immediately follow this tag. The `dlistDef` tag element specifies a term definition and follows one or more `dlistTerm` elements.

The content model for term definitions is quite broad, including paragraphs, lists, preformatted text, forms, tables, figures, and admonishments. Headers are not permitted, although implementers of HTML 3.0 user agents are advised to allow for this possibility in order to handle badly formed legacy documents.

**Syntax**

```
http.dlistDef (ctext, cclear, cattributes);
```

**Attributes**

```
ID, LANG, CLASS,
```

**Generates**

```
<DD CLEAR="cclear" cattributes>ctext
```

***htp.dlistOpen***

Prints an HTML tag that starts a definition list.

A *definition list* is a list of terms and corresponding definitions. Definition lists are typically formatted with the term on the left and the definition following on the right or on the next line. The definition text is typically indented with respect to the term.

An alternative format aligns the term to the left within a wide margin and places the definition on one or more lines to the right of the term. If the DT term does not fit in the DT column (one third of the display area), it may be extended across the page with the DD definition section moved to the next line, or the term may be wrapped onto successive lines of the left column. The opening list tag must be <DL>. It is followed by an optional list header (<LH>caption</LH>) and then by term names (<DT>) and definitions (<DD>).

**Syntax**

```
htp.dlistOpen (cclear, cattributes);
```

**Attributes**

```
ID, LANG, CLASS, COMPACT
```

**Generates**

```
<DL CLEAR="cclear" cattributes>
```

***htp.dlistClose***

Prints an HTML tag that ends a definition list.

**Syntax**

```
http.dlistClose
```

**Generates**

```
</DL>
```

***http.dlistTerm***

Prints an HTML tag used to insert the definition term inside the definition list. This tag must immediately follow the `http.dlistDef`. The `dlistTerm` tag element specifies a term name. You can have several terms per `dlistDef` element.

Term names are restricted to character-level markup only, including emphasis, inline images, and footnotes. Paragraph tags and other block-like elements such as headers are not permitted, although implementers of HTML 3.0 user agents are advised to allow for this possibility in order to handle badly formed legacy documents.

**Syntax**

```
http.dlistTerm (ctext, cclear, cattributes);
```

**Attributes**

```
ID, LANG, CLASS
```

**Generates**

```
<DT CLEAR="cclear" cattributes>ctext
```

***http.listHeader***

Prints an HTML tag at the beginning of the list. The `listHeader` tag `<LH>` simply displays a heading at the top of the list. List headers were added as of HTML 3.0. HTML 2.0 did not support list headers so developers typically used headers instead. The problem with using headers is one developer may choose to use `<H5>` while another may use `<H6>` so the lists no longer look uniform.

The opening list tag must be one of the appropriate list opening tags (<UL> (ulistOpen), <DIR> (dirlistOpen), <MENU> (menulistOpen)). It is followed by an optional list header (<LH>caption</LH>) and then by the first list item (<LI>).

### Syntax

```
http.listHeader (ctext, cattributes);
```

### Generates

```
<LH cattributes>ctext</LH>
```

### *http.listItem*

Prints an HTML tag that formats a listed item. This is used with ordered lists, (<OL> or http.olistOpen), unordered lists (<UL> or http.ulistOpen), menu lists (<MENU> or http.menulistOpen), or directory lists (<DIR> or http.dirlistOpen).

### Syntax

```
http.listItem (ctext, cclear, cdingbat, csrc, cattributes);
```

### Attributes

```
ID, LANG, CLASS, MD, SKIP, TYPE, VALUE
```

### Generates

```
<LI CLEAR="cclear" DINGBAT="cdingbat" SRC="csrc" cattributes>ctext
```

### *http.menulistOpen*

Prints an HTML tag that begins a list displaying one item per line, causing the list to appear more compact than an unordered list. The http.listItem follows the http.menulistOpen tag.

This element was superseded by extensions to the UL element. User agents are advised to continue to support it for the sake of legacy documents. Similar to UL, MENU consists of one or more LI elements. MENU lists are typically rendered without bullets in a more compact style than UL. You can get the same effect with <UL PLAIN> or http.ulistOpen(cattributes => 'PLAIN');

**Syntax**

```
http.menuListOpen;
```

**Generates**

```
<MENU>
```

**Example**

See the example at the beginning of this chapter.

***http.menuListClose***

Prints an HTML tag that ends a menu list.

**Syntax**

```
http.menuListClose;
```

**Generates**

```
</MENU>
```

***http.olistOpen***

Prints an HTML tag that is used to open an ordered list that presents items preceded by numbers.

HTML 3.0 gives you the ability to control the sequence number—to continue where the previous list left off or to start at a particular number. The numbering style is determined by associated style sheets (e.g., whether nested lists contribute to a compound item number such as 3.1.5, or whether numbers are rendered as Arabic, uppercase or lowercase Roman numerals, or by using the numbering scheme appropriate to the language context).

**Syntax**

```
http.olistOpen (cclear, cwrap, cattributes);
```

**Attributes**

```
ID, LANG, CLASS, CONTINUE, SEQNUM, COMPACT, TYPE, START
```

**Generates**

```
<OL CLEAR="cclear" WRAP="cwrap" cattributes>
```

**Example**

See the example at the beginning of this appendix.

***htp.olistClose***

Prints an HTML tag that ends an ordered list.

**Syntax**

```
htp.olistClose;
```

**Generates**

```
</OL>
```

***htp.ulistOpen***

Prints an HTML tag that is used to open an unordered (or bulleted) list.

HTML 3.0 gives you the ability to customize the bullets, to create the list without bullets, and to wrap list items horizontally or vertically for multicolumn lists.

**Syntax**

```
htp.ulistOpen (cclear, cwrap, cdingbat, csrc, cattributes);
```

**Attributes**

```
ID, LANG, CLASS, NOWRAP, PLAIN, MD, COMPACT, TYPE
```

**Generates**

```
<UL CLEAR="cclear" WRAP="cwrap" DINGBAT="cdingbat" SRC="csrc" cattributes>
```

***htp.ulistClose***

Prints an HTML tag that ends the unordered list.

**Syntax**

```
http.ulistClose;
```

**Generates**

```
</UL>
```

**Table Tags**

Table tags allow the user to insert tables and manipulate the size and columns of a table in a document.

***http.tableOpen***

Prints an HTML tag that begins an HTML table. The HTML table model has been chosen for its simplicity and flexibility. By default, the table is automatically sized according to the cell contents and the current window size. The COLSPEC attribute can be used when needed to exert control over column widths, either by setting explicit widths or by specifying relative widths. You can also specify the table width explicitly or as a fraction of the current margins (see the WIDTH attribute).

Tables start with an optional caption followed by one or more rows. Each row is formed by one or more cells, which are differentiated into header and data cells. Cells can be merged across rows and columns, and can include attributes that assist in rendering the table to speech and braille or for exporting table data into databases. The model provides little direct support for control over appearance—for example, border styles and margins—because these can be handled via subclassing and associated style sheets.

Tables can contain a wide range of content, such as headers, lists, paragraphs, forms, figures, preformatted text, and even nested tables. When the table is flush left or right, subsequent elements flow around the table if there is sufficient room. This behavior is disabled when the noflow attribute is given or the table align attribute is center (the default) or justify.

**Syntax**

```
http.tableOpen (cborder, calign, cnowrap, cclear, cattributes);
```

**Attributes**

ID, LANG, CLASS, NOFLOW, UNITS, COLSPEC, DP, WIDTH, CELLSPACING, CELLPADDING, FRAME, FLOAT, RULES

**Generates**

```
<TABLE "cborder" NOWRAP ALIGN="calign" CLEAR="cclear" cattributes>
```

***http.tableClose***

Prints an HTML tag that ends an HTML table. If you forget to close a table, you may get some strange results—most notably the failure of data to display in the browser even though it shows up in the HTML source.

**Syntax**

```
http.tableClose;
```

**Generates**

```
</TABLE>
```

***http.tableCaption***

Prints an HTML tag that places a caption in the inserted table. The CAPTION element is used to label a table or figure. Use the align attribute to specify the position of the caption relative to the table/figure. For example:

```
<CAPTION ALIGN=LEFT>Consulting Services Provided</CAPTION>
```

**Syntax**

```
http.tableCaption (ccaption, calign, cattributes);
```

**Attributes**

ID, LANG, CLASS

**Generates**

```
<CAPTION ALIGN="calign" cattributes>ccaption</CAPTION>
```

## ***http.tableData***

Prints an HTML tag that inserts data into the rows and columns of a selected table. See `http.tableHeader` for additional information.

### **Syntax**

```
http.tableData (cvalue, calign, cdp, crowspan, ccolspan, cnowrap,  
cattributes);
```

### **Attributes**

ID, LANG, CLASS, VALIGN, AXIS, AXES, WIDTH, BGCOLOR

### **Generates**

```
<TD ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP  
cattributes>cvalue</TD>
```

## ***http.tableHeader***

Prints an HTML tag that inserts a table header.

The TH and TD elements are used for table cells. TH is used for table header cells while TD is used for table data cells. This distinction gives user agents a means to render such cells distinctly by using a larger or heavier font for header cells. It is also needed when rendering to speech. The CLASS attribute can also be used to create heads and subheads. Together, these elements can be used together with style sheets to control the cell border style, and to fill color, etc.

The horizontal and vertical alignment of cell content is determined by the ALIGN and VALIGN attributes, respectively. In their absence, the alignment is inherited from the TR element for the row. The COLSPEC attribute of the enclosing TABLE element provides a convenient way of specifying the default horizontal alignment for columns.

The AXIS and AXES attributes can be used when rendering to speech to provide abbreviated names for each cell's header. Another application is to process table contents to be entered into a database sometime in the future. These attributes are then used to

assign database field names. The table's class attribute is used to tell the software which tables can be treated in this way.

### Syntax

```
http.tableHeader (cvalue, calign, cdp, cnowrap, crowspan, ccolspan,  
cattributes);
```

### Attributes

ID, LANG, CLASS, VALIGN, AXIS, AXES, WIDTH, BGCOLOR

### Generates

```
<TH ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan" NOWRAP  
cattributes>cvalue</TH>
```

### *http.tableRowOpen*

Prints an HTML tag that inserts a row tag into a table. The TR element acts as a container for a row of table cells defined with the TH or TD elements. You can set default horizontal and vertical alignment of cell contents for the row. You also have the ability to disable word wrap for the row, and thereafter use the <BR> element to determine line breaks and hence cell widths.

To assist with formatting tables for paged media, authors can indicate the leading and trailing rows for duplication when tables split across page boundaries. The recommended approach is to subclass rows using the CLASS attribute.

### Syntax

```
http.tableRowOpen (calign, cvalign, cdp, cnowrap, cattributes);
```

### Attributes

ID, LANG, CLASS

### Generates

```
<TR ALIGN="calign" VALIGN="cvalign" DP="cdp" NOWRAP cattributes>
```

## ***http.tableRowClose***

Prints an HTML tag that ends a row in a table.

### **Syntax**

```
http.tableRowClose;
```

### **Generates**

```
</TR>
```

## **Form Tags**

Form tags are used to create and manipulate an HTML form. Forms allow interactive data exchange between a web browser and a server program. In the case of the Oracle Application Server, forms allow you to prompt for input from the user and then insert or update data in the database. HTML forms are similar to the functionality available in Oracle SQL\*Forms 3.0 running in block mode, meaning that standard HTML forms allow the user to enter a screen of information and then send the entire set of information to the server. To provide users with the GUI functionality that they are accustomed to requires that you redesign the way you create your forms and use JavaScript. The syntax, parameter, and attribute information for the standard HTML form tags that can be categorized into the following tag types:

- **Input** Used for a large variety of types of input fields (e.g., single-line text, single-line password fields, checkboxes, radio buttons, and submit buttons).
- **Select** Used to allow the user to choose one or more of a set of alternatives described by textual labels. Usually rendered as a pull-down, pop-up, or fixed-size list.
- **Text area** Used to create a multiline input field.

## ***http.formOpen***

Prints an HTML tag that starts the form. The curl value (action) is required and is the URL of the CGI script or cartridge (usually the PL/SQL cartridge) to which the

contents of the form is sent. The method is either GET or POST, which are briefly discussed later in this appendix.

HTML forms can be used for questionnaires, hotel reservations, order forms, data entry, and a wide variety of other applications. The form is specified as part of an HTML document. The user fills in the form and then submits it. The user agent then sends the form's contents as designated by the FORM element. Typically, the contents are sent to an HTTP server, such as the Oracle Application Server, but you can also e-mail form contents for asynchronous processing via a CGI script.

Forms are created by placing input fields within paragraphs, preformatted text, lists, and tables. This gives considerable flexibility in designing the layout of forms.

HTML 3.0 supports the following kinds of fields:

- Simple text fields
- Multiline text fields
- Radio buttons
- Checkboxes
- Range controls (sliders or knobs)
- Single or multiple choice menus
- Scribble on image
- File widgets for attaching files to forms
- Submit buttons for sending form contents
- Reset buttons for resetting fields to their initial values
- Hidden fields for bookkeeping information
- Callouts
  - Simple text fields
  - Radio buttons
  - Submit and Reset buttons

- TEXTAREA field for multiline entry

### Syntax

```
http.formOpen (curl, cmethod, ctarget, cenctype, cattributes);
```

### Attributes

SCRIPT

### Generates

```
<FORM ACTION="curl" METHOD="cmethod" TARGET="ctarget" ENCTYPE="cenctype"  
attributes>
```

### *http.formClose*

Prints an HTML tag that closes the <FORM> tag

### Syntax

```
http.formClose;
```

### Generates

```
</FORM>
```

### *http.formCheckbox*

Prints an HTML tag that inserts a checkbox a user can toggle off or on. A checkbox field has two states: selected and unselected. Its name/value pair appears in the submitted data only when selected. Checkboxes are used for Boolean attributes. They can also be used for attributes that can take on multiple values at the same time. This is represented by a checkbox for each optional value, with the same name for each of the checkboxes. Unselected checkboxes don't appear in the submitted data. Therefore, if none of the checkboxes is selected, the parameter is not sent to PL/SQL. Both NAME and VALUE are required for checkboxes. To initialize the checkbox to its selected state, include the CHECKED attribute. Checkboxes provide an alternate method to using the SELECT element for multiple choice menus.

**Syntax**

```
http.formCheckbox (cname, cvalue, cchecked, cattributes);
```

**Attributes**

ID, LANG, CLASS, DISABLED, ERROR, MD

**Generates**

```
<INPUT TYPE="checkbox" NAME="cname" VALUE="cvalue" CHECKED cattributes>  
<Command>
```

***http.formHidden***

Prints an HTML tag that sends the content of a field along with a submitted form to the CGI program or the PL/SQL cartridge that is specified in form ACTION (curl) attribute. The field is not visible to the end user.

Hidden fields may be used to transmit state information about client/server interaction—for instance, a transaction identifier (such as the session ID). Depending on how you develop your application, these hidden fields may be needed so you can pass the correct session information to the next procedure. In other words, HTTP servers, such as Oracle Application Server, don't preserve state information from one request to the next, so you can use the hidden fields to pass this state information from one procedure to the next. See Chapter 16 for more information on security and passing such state information.

**Syntax**

```
http.formHidden (cname, cvalue, cattributes);
```

**Attributes**

ID, LANG, CLASS, DISABLED, ERROR, MD

**Generates**

```
<INPUT TYPE="hidden" NAME="cname" VALUE="cvalue" cattributes>
```

## ***http.formImage***

Prints an HTML tag that creates an image field that can be clicked causing the form to be immediately submitted. The coordinates of the selected point are measured in pixels and returned (along with other contents of the form) in two name/value pairs. If you name the image (e.g., `<form type=image name=myimage src=/backvid.gif>`), then the x-coordinate is submitted under the name of the field with ".x" appended (e.g., `myimage.x`), and the y-coordinate with the ".y" appended (e.g., `myimage.y`). If you do not name the image (e.g., `<form type=image src=/backvid.gif>`), then the x-coordinate is submitted under the name of "x", and the y-coordinate under the name of "y". If you have two unnamed images, it will only pass one x and one y parameter, but you won't know which image the parameters belong to because any value attribute is ignored. The image itself is specified by the CSRC attribute. These act like Submit buttons but include the location where the user clicked the image. The image is specified with the SRC attribute.

### **Syntax**

```
http.formImage (cname, csrc, calign, cattributes);
```

### **Attributes**

ID, LANG, CLASS, DISABLED, ERROR, MD

### **Generates**

```
<INPUT TYPE="image" NAME="cname" SRC="csrc" ALIGN="calign" cattributes>
```

## ***http.formPassword***

Prints an HTML tag that creates a single-line text entry field, but in this case the text is not displayed as it is entered. The built-in cut text (CTRL-X) and copy text (CTRL-C) keys that normally work in the Windows environment are disabled when you are in a password form field. This prevents a user from copying a password that was typed and left visible on a user's screen. However, if a default password is passed into the password field, a user can simply use the view source option to see the default password. Be aware that if you use the GET method for your form, the password field is passed as a text field on URL line and is readily available for anyone to see. When the user enters a password,

each character typed is echoed by a shadow character (e.g., an asterisk or the spaceband character). The user can see how many characters that have been typed but not what was typed.

### Syntax

```
http.formPassword (cname, csize, cmaxlength, cvalue, cattributes);
```

### Attributes

ID, LANG, CLASS, DISABLED, ERROR, MD

### Generates

```
<INPUT TYPE="password" NAME="cname" SIZE="csize" MAXLENGTH="cmmaxlength"  
VALUE="cvalue" cattributes>
```

## *http.formRadio*

Prints an HTML tag that inserts a radio button on the HTML form. This is used to create a set of radio buttons, each representing a different value. Only one of the buttons in a group can be toggled by the user. Each radio button field in a group should have the same name. Only the selected radio button generates a name/value pair in submitted data area. This requires an explicit VALUE attribute.

It's suitable for attributes that can take a single value from a set of alternatives. All radio buttons in the same group should be given the same NAME, which makes them mutually exclusive (only one of the buttons in the group is selected at a time). Only the selected radio button in the group generates a name/value pair in the submitted data. Both NAME and VALUE are required for radio buttons. To initialize the radio button to its selected state, include the CHECKED attribute. Radio buttons offer an alternative to using the SELECT element for single choice menus.

### Syntax

```
http.formRadio (cname, cvalue, cchecked, cattributes);
```

### Attributes

ID, LANG, CLASS, DISABLED, ERROR, MD

**Generates**

```
<INPUT TYPE="radio" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

***htp.formReset***

Prints an HTML tag that creates a RESET button that, when selected, resets all the form fields to their initial default values. The label to be displayed on the button may be specified, or the SRC attribute can be used to specify a graphic.

**Syntax**

```
htp.formReset (cvalue, cattributes);
```

**Attributes**

SRC, ALIGN, ID, LANG, CLASS, DISABLED, ERROR, MD

**Generates**

```
<INPUT TYPE="reset" VALUE="cvalue" cattributes>
```

***htp.formSubmit***

Prints an HTML tag that creates a button that, when selected, submits the form. When a Submit button is selected to submit a form and that button has a name attribute specified, the Submit button contributes a name/value pair to the submitted data. You can use the VALUE attribute to provide an uneditable label to be displayed on the button. The default label is browser-specific but is usually “Submit”. A graphic can be specified for the Submit button using the SRC attribute.

The Submit button normally makes no contribution to the submitted data. The exception is when the field includes a NAME attribute, in which case, the name and value attributes are included with the submitted data. This can be used when you need more than one Submit button on a form, each button having unique functionality associated with them. The NAME distinguishes which Submit button the user pressed. The Submit button is passed to PL/SQL like NAME=VALUE of button pressed. For example, if you had two buttons with a name of “Submit,” and one had a value of “Insert Duplicate” and the other had a value of “Update”, if the user pressed the “Insert

Duplicate” button, the browser would send this as `SUBMIT=Insert+Duplicate`. Note that your procedure must have an input parameter called “SUBMIT” and must also perform the correct functionality based on the value passed to it.

### Syntax

```
http.formSubmit (cname, cvalue, cattributes);
```

### Attributes

SRC, ALIGN, ID, LANG, CLASS, DISABLED, ERROR, MD

### Generates

```
<INPUT TYPE="submit" NAME="cname" VALUE="cvalue" cattributes>
```

## *http.formText*

Prints an HTML tag that creates a field for a single line of text. These are used for entering short text strings, such as numbers, dates, and the names of people. The visible width of the field in characters can be set with the `SIZE` attribute. When using a variable pitch font, the `SIZE` attribute sets the width in *em* units (which are half the point size). The user should be able to enter more than this, with the contents of the field scrolling horizontally as needed. The `MAXLENGTH` attribute can be used to specify the maximum number of characters permitted for the string.

If the `TYPE` attribute is missing, the `INPUT` element is assumed to be a single-line text field. The `NAME` attribute is used to identify the field when the form’s contents are converted to the name/value list. The `VALUE` field can be used to initialize the text string. Character entities can be used to include accented characters in this string.

### Syntax

```
http.formText (cname, csize, cmaxlength, cvalue, cattributes);
```

### Attributes

ID, LANG, CLASS, DISABLED, ERROR, MD

**Generates**

```
<INPUT TYPE="text" NAME="cname" SIZE="csize" MAXLENGTH="cmaxlength"  
VALUE="cvalue" cattributes>
```

**SELECT**

The SELECT form tags allows a drop-down list box (or fixed-size list box), listing specific values that the user can pick from. Multiple values can also be selected (if you allow the user to do so). Therefore, the SELECT Form tags perform functionality similar to a radio button (if MULTIPLE is not in the attributes, the field will be mutually exclusive) or a group of checkboxes (if MULTIPLE is in the attributes).

***http.formSelectOpen***

Prints an HTML tag that begins a select list of alternatives, and contains the attribute NAME, which specifies the name that will be submitted as a name/value pair.

The SELECT element is used for single- and multiple-choice menus. It is generally rendered as a drop-down or pop-up menu and offers a more compact alternative to using radio buttons for single-choice menus, or checkboxes for multiple-choice menus.

**Syntax**

```
http.formSelectOpen (cname, cprompt, nsize, cattributes);
```

**Attributes**

MULTIPLE, ID, LANG, CLASS, DISABLED, ERROR, SRC, MD, WIDTH, HEIGHT, UNITS, ALIGN

**Generates**

```
cprompt <SELECT NAME="cname" PROMPT="cprompt" SIZE="nsize" cattributes>
```

***http.formSelectOption***

Prints an HTML tag that represents one choice in the SELECT element. See http.formSelectOpen example for more information.

When the form is submitted, the **NAME** of the enclosing **SELECT** element is paired with the **OPTION**'s **VALUE** attribute to contribute a name/value pair for the selection. Unselected options don't contribute to the form's submitted data. You can initialize the option to its selected state by including the **SELECT** attribute.

The **OPTION** element can occur only within a **SELECT** element. It represents a possible choice. It can contain only text, together with **SGML** entities for accented characters, and so on.

The **SHAPE** attribute is used for graphical menus to specify the region of the background image to be associated with this option. It uses the same definition as that used by the **anchor** element.

### Syntax

```
http.formSelectOption (cvalue, cselected, cattributes);
```

### Attributes

ID, LANG, CLASS, DISABLED, ERROR, SHAPE,

### Generates

```
<OPTION SELECTED cattributes>cvalue
```

### *http.formSelectClose*

Prints an HTML tag that ends a select list of alternatives.

### Syntax

```
http.formSelectClose;
```

### Generates

```
</SELECT>
```

### Text Area

The following form tags are used to support text input via forms.

## ***http.formTextarea***

Prints an HTML tag that creates a text field that has no predefined text in the text area. This tag is used to enable the user to enter several lines of text.

### **Syntax**

```
http.formTextarea (cname, nrows, ncolumns, calign, cattributes);
```

### **Attributes**

ID, LANG, CLASS, DISABLED, ERROR

### **Generates**

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign"  
cattributes></TEXTAREA>
```

## ***http.formTextarea2***

Prints an HTML tag that creates a text field that has no predefined text in the text area. It's used to enable the user to enter several lines of text. The difference between this and `formTextarea` is the `cwrap` parameter, which specifies a wrap style. See `http.formTextarea` for more HTML information.

### **Syntax**

```
http.formTextarea2 (cname, nrows, ncolumns, calign, cwrap, cattributes);
```

### **Attributes**

ID, LANG, CLASS, DISABLED, ERROR

### **Generates**

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign"  
WRAP="cwrap" cattributes></TEXTAREA>
```

### ***http.formTextareaOpen***

Prints an HTML tag that opens a text area where you can insert predefined text that will always appear in the text field. See `http.formTextarea` for more HTML information.

#### **Syntax**

```
http.formTextareaOpen (cname, nrows, ncolumns, calign, cattributes);
```

#### **Attributes**

ID, LANG, CLASS, DISABLED, ERROR

#### **Generates**

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign"  
cattributes>
```

### ***http.formTextareaOpen2***

Prints an HTML tag that opens a text area where you can insert predefined text that will always appear in the text field. The difference between this and `formTextareaOpen` is the `cwrap` parameter, which specifies a wrap style. See `http.formTextarea` for more HTML information.

#### **Syntax**

```
http.formTextareaOpen (cname, nrows, ncolumns, calign, cwrap, cattributes);
```

#### **Attributes**

ID, LANG, CLASS, DISABLED, ERROR

#### **Generates**

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign" WRAP =  
"cwrap" cattributes>
```

### ***http.formTextareaClose***

Prints an HTML tag that ends the `TextArea` field.

## Syntax

```
http.formTextareaClose;
```

## Generates

```
</TEXTAREA>
```

## Frame Tags

Frames enable you to create web pages that contain multiple windows. Netscape Navigator 2.0+ and Internet Explorer 3.0+ support frames, but they are not an HTML 3.0 standard. The <FRAMESET> tag allows you to specify a section of the page to be used by frame-enabled browsers, while the text within the <NOFRAME> tag allows you to specify how the page will look in browsers that cannot use frames. You have to determine whether you want to support frameless browsers. If you decide you want to support them, doing so can require two sets of code and makes for dual maintenance. Frames can be nested within each other.

### *http.framesetOpen*

Prints an HTML tag to open a container of web page frames. Framesets can be nested within each other. The <FRAMESET> tag identifies the number of rows and columns on the page. Then within each box or frame you can specify more framesets to nest or subdivide a frame. The <FRAME> tag is used to place information in each frame, which is typically done by referencing another URL. The <FRAMESET> tag, like the <A>nchor and <FORM> tags, can use an attribute called TARGET to indicate in which frame the data created as the result of an action should be placed. The first parameter of the framesetOpen command specifies the rows, and the second specifies the columns; you can specify only one parameter at a time, and your page can be broken up into either rows or columns by each <FRAMESET> tag. Frames can be subdivided thereafter.

You can use the <FRAMESET> tag to size the columns and rows of a frame in the following ways:

- As an absolute value in pixels
- As a relative value in percentage of window width

- As a relative value following the CALS table model syntax

### ***http.framesetClose***

Prints an HTML tag to close a container of web page frames.

#### **Syntax**

```
http.framesetClose;
```

#### **Generates**

```
</FRAMESET>
```

### ***http.frame***

Prints an HTML tag that defines a single frame within a frameset. For each frame, you define the URL (i.e., the source) for the frame and its key characteristics.

The only tag that you can place within the beginning and ending <FRAMESET> tags is the <FRAME> tag. The <FRAME> tag is used to specify the source for the frame and to set key characteristics of the frame (e.g., the name of the frame name, its margins, whether or not scrollbars or resizing are allowed, and so on). Frames are filled in the order in which they are created. For example, in the above <FRAMESET> example, the first row (which is 15 percent of the page) is filled with a frame that contains the source to pull the logo at the top of the page. Within the second row (which is 85 percent of the page) is filled with another frame set with two columns of 10 percent and 90 percent. The two columns in the bottom row are then filled with the respective source data.

#### **Syntax**

```
http.frame(csrc, cname, cmarginwidth, cmarginheight, cscrolling, cnoresize,  
cattributes);
```

#### **Generates**

```
<FRAME SRC="csrc" NAME="cname" MARGINWIDTH="cmarginwidth"  
MARGINHEIGHT="cmarginheight" SCROLLING="cscrolling" NORESIZE cattributes>
```

### ***http.noframesOpen***

Prints an HTML tag to open a container of content, which is viewable by web browsers that are not frame-enabled.

All of the text between this tag and the closing tag will be used by a browser that does not support frames. When designing your web applications, you must decide if you're going to support browsers that don't support forms.

#### **Syntax**

```
http.noframesOpen;
```

#### **Generates**

```
<NOFRAMES>
```

### ***http.noframesClose***

Prints an HTML tag to close a container of content, which is viewable by web browsers that are not frame-enabled.

#### **Syntax**

```
http.noframesClose;
```

#### **Generates**

```
</NOFRAMES>
```

## **General Format Tags**

General format tags are used to control the hierarchical structure of your web pages. General format tags include the `http.div` procedure (also known as the DIV element) and the `http.header`.

### ***http.div***

Prints an HTML tag to create document divisions. The DIV element is used with the CLASS attribute to represent different kinds of containers (e.g., chapter, section, abstract, or appendix). As of HTML 3.0 there are no visible differences. However for

HTML 3.1, 4.0, and later versions, the `division` command does change the look and feel of your pages.

### Syntax

```
http.div(calign, cattributes);
```

### Attributes

ID, LANG, CLASS, NOWRAP, CLEAR

### Generates

```
<DIV ALIGN="calign" cattributes>
```

### *http.header*

Prints the HTML tag for a heading level with the value of the heading level assigned in the *nsiz*e parameter. Valid levels are 1 through 6, which are presented as H1, H2, H3, H4, H5, and H6, with H1 being the highest (or most important) level and H6 the lowest (or least important).

### Syntax

```
http.header (nsiz
```

e, cheader, calign, cnowrap, cclear, cattributes);

### Attributes

ID, LANG, CLASS, SEQNUM, SKIP, DINGBAT, SRC, MD

### Generates

```
<Hnsiz
```

e ALIGN="calign" NOWRAP CLEAR="cclear"  
cattributes>cheader</Hnsize>

### Lines

Line tags are used to control the display of horizontal rules that appear on your web pages. Line tags include `http.hr` and `http.line`.

### *http.hr*

Alias for `http.line`.

## ***http.line***

Prints the HTML tag that generates a line (or horizontal rule) in the HTML document. The CSRC attribute enables you to specify a custom image as the source of the line.

The line element is used for horizontal rules that act as dividers between sections. The CSRC attribute can be used to designate a custom graphic.

### **Syntax**

```
| http.line (cclear, csrc, cattributes);
```

### **Attributes**

ID, CLASS, MD, SIZE, WIDTH, ALIGN, NOSHADE

### **Generates**

```
| <HR CLEAR="cclear" SRC="csrc" cattributes>
```

## **Breaks and No Breaks**

Line breaks are a great way to control the flow of text. Break tags include

- `http.br`
- `http.nl`
- `http.nobr`
- `http.wbr`

### ***http.br***

Alias for `http.nl`.

### ***http.nl***

Prints the HTML tag that inserts a new line. Line break and tab elements can be used when you need a little more control over how the browser renders the text. By default, browsers ignore line breaks in HTML files. Instead they insert soft line breaks to word wrap lines of text. The `<BR>` element is used to force a line break.

**Syntax**

```
http.nl (cclear, cattributes);
```

**Attributes**

ID, LANG, CLASS

**Generates**

```
<BR CLEAR="cclear" cattributes>
```

***http.nobr***

Prints an HTML tag to turn off line-breaking with a section of text.

**Syntax**

```
http.nobr (ctext);
```

**Generates**

```
<NOBR>ctext</NOBR>
```

***http.wbr***

Prints an HTML tag to insert a soft line break within a section of NOBR text. See `http.nobr` for more information and an example.

**Syntax**

```
http.wbr;
```

**Generates**

```
<WBR>
```

**Paragraphs**

The paragraph tags `http.para` and `http.paragraph` give you the ability to insert new lines with extra space between paragraphs. Note that there is no toolkit “end paragraph” command, but there is an end paragraph command for HTML, `</P>`. Including the end

paragraph command in your output changes the page's presentation by adding an extra space between paragraphs.

### ***htp.para***

Prints an HTML tag that indicates that the text previous to it should be formatted as a paragraph. For consistency, you might as well use `htp.paragraph` without parameters, since it defaults to NULL.

#### **Syntax**

```
| htp.para
```

#### **Generates**

```
| <P>
```

### ***htp.paragraph***

Prints the same HTML tag as `htp.para` except that parameters pass in exact alignment, leading, wrapping, and attributes.

The `<P>` tag is used to define a paragraph. The exact rendering (i.e., indentation, leading, etc.) is not defined and may be a function of other tags, style sheets, etc. The `ALIGN` attribute can be used to specify the horizontal alignment. Paragraph elements have the same content model as headers, which means that paragraphs can include text and character-level markup, such as character emphasis, inline images, form fields, and math.

#### **Syntax**

```
| htp.paragraph (calign, cnowrap, cclear, cattributes);
```

#### **Attributes**

ID, LANG

#### **Generates**

```
| <P ALIGN="calign" NOWRAP CLEAR="cclear" cattributes>
```

## Formatting Text and Images

Formatting tags let you control the alignment of the text and images that appear on your web pages.

### *http.address*

Prints an HTML tag that lets you specify the address, author, and signature of document. The ADDRESS element specifies the address, signature, and author of the current document. It is typically placed at the top or bottom of the document and can be placed outside of the BODY section.

#### Syntax

```
http.address (cvalue, cnowrap, cclear, cattributes);
```

#### Attributes

ID, LANG, CLASS

#### Generates

```
<ADDRESS CLEAR="cclear" NOWRAP cattributes>cvalue</ADDRESS>
```

### *http.blockquoteOpen*

Prints an HTML tag that precedes a paragraph of quoted text. The BLOCKQUOTE tag is used for extended quotations. As of HTML 3.0, the tag name has been abbreviated from HTML 2.0's BLOCKQUOTE to the more convenient BQ, and the content model has been extended to allow the source of the quotation to be credited.

#### Syntax

```
http.blockquoteOpen (cnowrap, cclear, cattributes);
```

#### Attributes

ID, LANG, CLASS

#### Generates

```
<BLOCKQUOTE CLEAR="cclear" NOWRAP cattributes>
```

***htp.blockquoteClose***

Prints an HTML tag that ends the <BLOCKQUOTE> section of quoted text.

**Syntax**

```
htp.blockquoteClose;
```

**Generates**

```
</BLOCKQUOTE>
```

***htp.center***

Prints a pair of HTML tags to center a section of text (or image) within a web page.

**Syntax**

```
htp.center(ctext);
```

**Generates**

```
<CENTER>ctext</CENTER>
```

***htp.centerOpen***

Prints an HTML tag to open a centered section of text (or image) within a web page. Text (or images) can be centered with the <CENTER> tag or with the ALIGN=CENTER attribute. The <CENTER> tag enables you to center portions of pages by putting the <CENTER> tag before the text (or image) you want centered and the </CENTER> tag after the text (or image).

**Syntax**

```
htp.centerOpen;
```

**Generates**

```
<CENTER>
```

### ***http.centerClose***

Prints an HTML tag to close a centered section of text within a web page.

#### **Syntax**

```
| http.centerClose;
```

#### **Generates**

```
| </CENTER>
```

### ***http.listingOpen***

Prints an HTML tag to indicate the beginning of fixed-width text in the body of an HTML page.

#### **Syntax**

```
| http.listingOpen;
```

#### **Generates**

```
| <LISTING>
```

### ***http.listingClose***

Prints an HTML tag to end the fixed-width section of text.

#### **Syntax**

```
| http.listingClose;
```

#### **Generates**

```
| </LISTING>
```

### ***http.preOpen***

Prints an HTML tag that indicates the beginning of preformatted text in the body of the HTML page. Preformatted text between the start and end PRE tags is rendered using a fixed-width font; in addition, whitespace characters are treated literally, adhering to the following conditions:

- The spacing and line breaks are rendered directly, unlike other elements, for which repeated whitespace characters are collapsed to a single-space character and line breaks are automatically introduced.
- Line breaks within the text are rendered as a move to the beginning of the next line. The exceptions are line breaks immediately following the starting PRE tag or immediately preceding the ending PRE tag, which should be ignored.
- The <P> tag should be avoided, but for robustness, user agents are encouraged to treat these tags as line breaks.
- Anchor elements and character-highlighting elements may be used.
- FORM elements may be included and the fixed-width font exploited to control layout (the TAB or TABLE elements give similar control for normal text).
- Block-like elements such as headers, lists, figures, and tables should be avoided.
- The horizontal tab character (encoded in U.S. ASCII and ISO 8859-1 as decimal 9) should be interpreted as the smallest nonzero number of spaces that will leave the number of characters so far on the line as a multiple of eight. Its use is discouraged.

### Syntax

```
http.preOpen (cclear, cwidth, cattributes);
```

### Attributes

ID, LANG, CLASS

### Generates

```
<PRE CLEAR="cclear" WIDTH="cwidth" cattributes>
```

### *http.preClose*

Prints an HTML tag that ends the preformatted section of text.

### Syntax

```
http.preClose;
```

**Generates**

```
| </PRE>
```

**Character Format for Fonts**

Character format tags for fonts let you define the typeface, color, and size of the fonts used on your web pages.

***htp.basefont***

Prints an HTML tag that specifies the base font size for a web page. This sets the default size of the font for the current page. Most browsers allow users to set their default font (both proportional and fixed-width) sizes. Base fonts and font sizes are relative to the font established by the user as the default font. Keep in mind that if you use a relational font size and the user's default font size is 7, you may quickly render text unreadable for such users.

**Syntax**

```
| htp.basefont (nsize) ;
```

**Generates**

```
| <BASEFONT SIZE="nsize">
```

***htp.fontOpen***

Prints an HTML tag that indicates the beginning of a section of text with the specified font characteristics. This tag changes the size, color, typeface, and other attributes for the text enclosed within.

Note: This is a Netscape- and Internet Explorer–supported tag, but is not part of HTML 3.0.

**Syntax**

```
| htp.fontOpen(ccolor, cface, csize, cattributes);
```

**Generates**

```
<FONT COLOR="ccolor" FACE="cface" SIZE="csize" cattributes>
```

***htp.fontClose***

Prints an HTML tag that indicates the end of a section of text with the specified font characteristics.

**Syntax**

```
htp.fontClose;
```

**Generates**

```
</FONT>
```

**Special Character Format**

Special character format tags support superscript, subscript, plain text, strikethrough text, and some proportional sizing.

***htp.big***

Prints a pair of HTML tags that specify that the text they surround is rendered using a “big” font—“big” meaning large print. The <BIG> element specifies that the enclosed text should be displayed, if practical, using a large font (compared with the current font). This is a new command available in HTML 3.0.

**Syntax**

```
htp.big(ctext, cattributes);
```

**Generates**

```
<BIG cattributes>ctext</BIG>
```

***htp.dfn***

Prints a pair of HTML tags that specify the text they surround is rendered in italics. The <DFN> element indicates the defining instance of a term. This is a new tag available in HTML 3.0.

**Syntax**

```
http.dfn(ctext);
```

**Generates**

```
<DFN>ctext</DFN>
```

***http.plaintext***

Prints a pair of HTML tags that specify the text they surround be rendered with fixed-width type.

**Syntax**

```
http.plaintext(ctext, cattributes);
```

**Generates**

```
<PLAINTEXT cattributes>ctext</PLAINTEXT>
```

***http.s***

Prints a pair of HTML tags that specify the text they surround be rendered in strikethrough type. The <S> element specifies that the enclosed text should be displayed with a horizontal line running through the text. If this is not practical, an alternative mapping is allowed. This is a new command as of HTML 3.0.

**Syntax**

```
http.s(ctext, cattributes);
```

**Generates**

```
<S cattributes>ctext</S>
```

***http.small***

Prints a pair of HTML tags that specify the text they surround is rendered using a small font—“small” meaning small print. The <SMALL> element specifies that the enclosed text should be displayed, if practical, using a small font (compared with normal text). This is a new command as of HTML 3.0.

**Syntax**

```
http.small(ctext, cattributes);
```

**Generates**

```
<SMALL cattributes>ctext</SMALL>
```

***http.strike***

Prints a pair of HTML tags that specify the text they surround be rendered in strikethrough type.

**Syntax**

```
http.strike(ctext, cattributes);
```

**Generates**

```
<STRIKE cattributes>ctext</STRIKE>
```

***http.sub***

Prints a pair of HTML tags that specify the text they surround is rendered as a subscript. The <SUB> element specifies that the enclosed text should be displayed as a subscript, and if practical, using a smaller font (compared with normal text). The ALIGN attribute for SUB is only meaningful within the MATH element. This is a new command as of HTML 3.0. For integral and related signs the SUB and SUP elements are used for the lower and upper limits, respectively.

**Syntax**

```
http.sub(ctext, calign, cattributes);
```

**Generates**

```
<SUB ALIGN="calign" cattributes>ctext</SUB>
```

***http.sup***

Prints a pair of HTML tags that specify the text they surround is rendered as a superscript. The <SUP> element specifies that the enclosed text should be displayed as a

superscript, and if practical, using a smaller font (compared with normal text). The ALIGN attribute for SUP is only applicable within the MATH element. This is a new command as of HTML 3.0.

### Syntax

```
htp.sup(ctext, calign, cattributes);
```

### Generates

```
<SUP ALIGN="calign" cattributes>ctext</SUP>
```

## Character Format Tags

The character format tags are used to specify or alter the appearance of the marked text. Character format tags have opening and closing elements and affect only the text that they surround.

Character format tags give hints to the browser as to how a character or character string should appear, but each browser actually determines its appearance. Essentially, the browser places the text into categories. All the text in a given category is given the same special treatment determined by the browser. For example, the HTML string `<STRONG> <body copy> </STRONG>` might appear as bold in some browsers, and it might flash in others. If a specific text attribute, such as bold, is desired, a physical format tag may be necessary.

### *htp.cite*

Prints a pair of HTML tags that specify the text they surround as a citation. The text is usually rendered in italics.

### Syntax

```
htp.cite (ctext, cattributes);
```

### Generates

```
<CITE cattributes>ctext</CITE>
```

### ***htp.code***

Prints a pair of HTML tags that specify the text they surround as an example of code output. The text is usually rendered in a monospace format (e.g., Courier).

#### **Syntax**

```
htp.code (ctext, cattributes);
```

#### **Generates**

```
<CODE cattributes>ctext</CODE>
```

### ***htp.em***

Alias for `htp.emphasis`.

### ***htp.emphasis***

Prints a pair of HTML tags that specify the text they surround as requiring typographic emphasis. This tag is equivalent to `htp.em`. The text is usually rendered in italics.

#### **Syntax**

```
htp.emphasis (ctext, cattributes);
```

#### **Generates**

```
<EM cattributes>ctext</EM>
```

### ***htp.kbd***

Alias for `htp.keyboard`.

### ***htp.keyboard***

Prints a pair of HTML tags that specify the text they surround as text typed in by the user, which usually is rendered in a monospaced font. This tag is equivalent to `htp.kbd`.

**Syntax**

```
http.keyboard (ctext, cattributes);
```

**Generates**

```
<KBD cattributes>ctext</KBD>
```

***http.sample***

Prints a pair of HTML tags that specify the text they surround as a sequence of literal characters that must be typed in the exact sequence in which they appear. The text is usually rendered in a monospaced font.

**Syntax**

```
http.sample (ctext, cattributes);
```

**Generates**

```
<SAMP cattributes>ctext</SAMP>
```

***http.strong***

Prints a pair of HTML tags that specify the text they surround to be given strong typographic emphasis. The text is usually rendered in bold.

**Syntax**

```
http.strong (ctext, cattributes);
```

**Generates**

```
<STRONG cattributes>ctext</STRONG>
```

***http.variable***

Prints a pair of HTML tags that specify the text they surround as a variable name or a variable that might be entered by the user. The text is usually rendered in italics.

**Syntax**

```
http.variable (ctext, cattributes);
```

**Generates**

```
<VAR cattributes>ctext</VAR>
```

**Physical Format Tags**

The physical format tags are used to specify the format of the marked text.

***htp.bold***

Prints a pair of HTML tags that specify the text they surround is to be rendered in boldface.

**Syntax**

```
htp.bold (ctext, cattributes);
```

**Generates**

```
<B cattributes>ctext</B>
```

***htp.italic***

Prints a pair of HTML tags that specify the text they surround is to be rendered in italics.

**Syntax**

```
htp.italic (ctext, cattributes);
```

**Generates**

```
<I cattributes>ctext</I>
```

***htp.teletype***

Prints a pair of HTML tags that specify the text they surround is to be rendered in a fixed-width monospaced font (e.g., Courier).

**Syntax**

```
htp.teletype (ctext, cattributes);
```

**Generates**

```
<TT cattributes>ctext</TT>
```

**OWA Packages*****OWA\_COOKIE Package***

OWA\_COOKIE is a package that provides datatypes, procedures, and functions that enable you to send HTTP cookies to and from the client's browser. HTTP cookies are opaque strings sent to the browser to maintain the state between HTTP calls. State can be maintained throughout the client's session or longer if an expiration date is included. The system date is calculated with reference to the information specified in the OWA\_INIT package.

The best way to look at cookies is as though the client stores a set of variables for your programs that are running on the server. The variables serve to provide memory to the browser. Any time you lose contact with the client (which is after every request), the client can still provide that information. Cookies should not be used to track every variable. When a user is moving from page to page, hidden variables provide a good alternative to cookies, and they work great in forms. If you are building anchors, you can build them with a set of parameters attached right in the hypertext reference (HREF). Cookies should be used for variables (usually only one per application—`session_id`) that you want to retain beyond the life of the current session.

**Datatypes**

Since the HTTP standard is that cookie names can be overloaded—that is, multiple values can be associated with the same cookie name—this PL/SQL RECORD holds all the values associated with a given cookie name via the following fields:

name	varchar2(4000)
vals	vc_arr
numvals	integer

**owa\_cookie.send**

This is a procedure that transmits a cookie to the client. This procedure must occur in the context of an OWA procedure's HTTP header output.

**Syntax**

```
owa_cookie.send(name, value, expires, path, domain, secure)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
name	in	vvarchar2	No	Cookie name
value	in	vvarchar2	No	Value to set cookie to
expires	in	date	Yes, default is NULL	Date of expiration for this cookie. At this date and time, the browser will remove the cookie from its cookie list. Keep in mind that by default (if a cookie has no expiration date), cookies last for the duration of the user session. When the user exits from the browser, the cookies expire and are removed. If you want a cookie to last beyond the life of a session, you'll need to set an expiration date.
path	in	vvarchar2	Yes, default	The default path for a

			is NULL	<p>cookie is from the directory from which it came. This means that if you created the cookie from a procedure called <i>get_vendors</i> that was in a package called <i>oug</i> from an agent named <i>moug</i>, the path might look something like <i>www.tusc.com/moug/owa/oug.get_vendors</i>, which the path is <i>/moug/owa</i>. Therefore, this cookie by default would be available to any other procedures executed from the <i>moug</i> agent. However, if you had a virtual administration path called <i>moug-admin</i>, the cookie created from the path <i>moug/owa</i> would not be available to <i>moug-admin</i>. If you wanted the cookie to be available from your entire site, you would have to set the path to <i>/</i> (root).</p>
domain	in	varchar2	Yes, default	Domain that may access

			is NULL	<p>this cookie. By default the domain will be set to the host name of the web server that serves the page. You may set the domain to the domain that your server is on; you may not set it to another domain. For example, TUSC's domain is <i>tusc.com</i>; if the server servicing this web page were <i>resources.tusc.com</i>, we could set the domain to <i>tusc.com</i>, but by default it would be set to <i>resource.tusc.com</i>. We could <i>not</i> set the domain to <i>yahoo.com</i>.</p>
secure	in	varchar2	Yes, default is NULL	<p>Is this a secure cookie? By default, cookies are insecure, which means that they are transmitted over normal, insecure, HTTP connections. If a cookie is marked as secure, then it would be transmitted only when the browser and server are connected via HTTPS or</p>

				another secure protocol.
--	--	--	--	--------------------------

**Generates**

```
Set-Cookie: <name>=<value> expires=<expires> path=<path> domain=<domain>
secure
```

**owa\_cookie.get Function**

This function retrieves a cookie from the client into a variable defined as a datatype of owa\_cookie.cookie.

**Syntax**

```
owa_cookie.get (name)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
name	in	varchar2	No	Return the value of a specific cookie

**Generates**

```
Get-cookie: <name>
```

**owa\_cookie.get\_all**

This procedure returns all cookie name/value pairs from the client's browser that are associated with your domain.

**Syntax**

```
owa_cookie.get_all (names, vals, num_vals)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
-----------	-----------------------	----------	-----------	-------------

names	out	vc_arr	No	Array into which all cookie names are placed
vals	out	vc_arr	No	Array into which all cookie values are placed
num_vals	out	integer	No	The number of cookies returned

### Generates

Arrays of the names and of the values in the order received and the count of the combinations.

### owa\_cookie.remove

This procedure forces a cookie to expire immediately. The output of this procedure must be embedded in an HTML header.

Tip: Since the cookie is stored on the client's computer, the cookie is not actually deleted (or immediately expired) until the generated page is sent to the client. This does not occur until the procedure is complete.

### Syntax

```
owa_cookie.remove(name, value, path)
```

### Parameters

Parameter	In, Out, or In Out	Datatype	Optional?	Description
Name	in	vvarchar2	No	Name of cookie to be removed
value	in	vvarchar2	No	Value to set expired cookie to
path	in	vvarchar2	Yes,	Path of cookie

			default is NULL	
--	--	--	--------------------	--

**Generates**

```
Set-Cookie: <name>=<value> expires=01-JAN-1990 path=<path>
```

***OWA\_IMAGE Package***

This package contains datatypes and functions that you can use to get the coordinates of where the user clicked on an HTML form image. If you were to define an HTML form that contains only an image (no other form fields) named MY\_IMG, when the user clicks an image, HTML performs a form submit and passes the (x- and y-) coordinates indicating where the user clicked on the image.

**Datatypes**

point	Provides the x- and y-coordinates of where a user clicks on an image map
-------	--

**Package Variables**

Null_point	A variable used to default point parameters. x and y both are NULL.
------------	---

**owa\_image.get\_x**

This function produces the x-coordinate of the image map.

**Syntax**

```
owa_image.get_x(p)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
p	in	point	No	Point to produce an x-

				coordinate for the image map.
--	--	--	--	-------------------------------

**Generates**

x-coordinate as integer

**owa\_image.get\_y**

This function produces the y-coordinate of the image map.

**Syntax**

```
owa_image.get_Y(p)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
p	in	point	no	Point to produce a y-coordinate for the image map.

**Generates**

y-coordinate as integer

***OWA\_INIT Package***

This package contains functions and procedures that initialize the cartridge. It also provides constants that you override to set the time zone used by cookies. Cookies use expiration dates defined in Greenwich mean time (GMT). If you are not on GMT, you can specify your time zone using one of these two constants:

- If your time zone is recognized by Oracle, you can specify it directly using `dbms_server_timezone`:

```
dbms_server_timezone constant varchar2 := 'MST';
```

The value for this is a string abbreviation for your time zone.

- If your time zone is not recognized by Oracle, use `dbms_server_gmtdiff` to specify the offset of your time zone from GMT. Specify a positive number if your time zone is ahead of GMT. Use a negative number if your time zone is behind GMT:

```
dbms_server_gmtdiff    constant number := -4;
```

After making the appropriate changes, you'll need to reload the package.

### Constants

<code>dbms_server_timezone</code>	Text string abbreviation for your time zone.
<code>dbms_server_gmtdiff</code>	Number of hours your time zone diverges from GMT. Use a positive number if your time zone is ahead of GMT; otherwise, use a negative number.

### Authorize

In this section you'll find the default *OWA\_INIT* package body that is installed with Application Server. As you can see, the default security for the PL/SQL cartridge is set to *OWA\_SEC.NO\_CHECK*. In short, this means that you are required to develop your own security methods for each procedure called because there is no security by default. In addition, you'll note that a procedure in *OWA\_INIT* called *AUTHORIZE* will be called if you change the default security to *OWA\_SEC.GLOBAL*. In other words, if you set the parameter to *OWA\_SEC.GLOBAL*, the *OWA\_INIT.AUTHORIZE* function will be called whenever the PL/SQL cartridge is invoked.

### **OWA\_OPT\_LOCK Package**

This package contains functions and procedures that enable you to impose database optimistic locking strategies, so as to prevent lost updates.

Since HTTP is a stateless protocol, conventional database-locking schemes cannot be used directly. The *OWA\_OPT\_LOCK* package works around this by giving you a choice of two ways of dealing with the lost update problem. (The lost update problem occurs when a user selects and then attempts to update a row whose value has been changed in the meantime by another user.)

- **The Hidden Fields Method** This stores the previous values in hidden fields in the HTML page. When the update is performed, it checks these values against the current state of the database. This is implemented with the *owa\_opt\_lock.store\_values* procedure.
- **The Checksum Method** This stores a checksum rather than the values themselves. This is implemented with the *owa\_opt\_lock.checksum* function.

Both of these techniques are *optimistic*; that is, they do not prevent other users from performing updates but reject the current update if an intervening update has occurred.

### ***owa\_opt\_lock.vcArray* Datatype**

This datatype is a PL/SQL table intended to hold ROWIDs.

```
type vcArray is table of varchar2(2000) index by binary_integer
```

### ***owa\_opt\_lock.checksum* Function**

This function returns a checksum value for a specified string or for a row in a table. For a row in a table, the function calculates the checksum value based on the values of the columns in the row. This function comes in two versions:

- The first version returns a checksum based on the specified string. This is a “pure” 32-bit checksum executed by the database and based on the Internet 1 protocol.
- The second version returns a checksum based on the values of a row in a table. This is an “impure” 32-bit checksum based on the Internet 1 protocol.

### **Syntax**

```
owa_opt_lock.checksum(p_buff in varchar2) return number;  
  
owa_opt_lock.checksum(  
    p_owner      in   varchar2  
    p_tname      in   varchar2  
    p_rowid      in   rowid)  
    return number;
```

**Parameters**

- **p\_buff** The string for which you want to calculate the checksum
- **p\_owner** The owner of the table
- **p\_tname** The table name
- **p\_rowid** The row in p\_tname for which you want to calculate the checksum value.

You can use the `owa_opt_lock.get_rowid` function to convert `vcArray` values to proper rowids.

**Return Value**

A checksum value

***owa\_opt\_lock.get\_rowid Function***

This function returns the ROWID datatype from the specified `owa_opt_lock.vcArray` datatype.

**Syntax**

```
owa_opt_lock.get_rowid(p_old_values in vcArray) return rowid;
```

**Parameters**

- **p\_old\_values** This parameter is usually passed in from an HTML form.

**Return Value**

A ROWID

***owa\_opt\_lock.store\_values Procedure***

This procedure stores the column values of the row that you want to update later. The values are stored in hidden HTML form elements.

Before you update the row, you compare these values with the current row values to ensure that the values in the row have not been changed. If the values have been changed, you can warn the users, and let them decide if the update should still take place.

**Syntax**

```
owa_opt_lock.store_values (  
    p_owner          in   varchar2  
    p_tname          in   varchar2  
    p_rowid          in   rowid);
```

**Parameters**

- **p\_owner** The owner of the table
- **p\_tname** The name of the table
- **p\_rowid** The row for which you want to store values

**Generates**

A series of hidden form elements:

- One hidden form element is created for the table owner. The name of the element is *old\_p\_owner*, where *p\_owner* is the name of the table owner. The value of the element is the owner name.
- One hidden form element is created for the table name. The name of the element is *old\_p\_tname*, where *p\_tname* is the name of the table. The value of the element is the table name.
- One element is created for each column in the row. The name of the element is *old\_p\_rowid*, where *p\_rowid* is the rowid of the table record. The value of the element is the column value.

***owa\_opt\_lock.verify\_values Function***

This function verifies whether or not values in the specified row have been updated since the last query. This function is used with the *owa\_opt\_lock.store\_values* procedure.

**Syntax**

```
owa_opt_lock.verify_values (p_old_values in vcArray) return boolean;
```

## Parameters

A PL/SQL table containing the following information:

- **p\_old\_owner** Specifies the owner of the table
- **p\_old\_tname** Specifies the table
- **p\_old\_rowid** Specifies the rowid of the row you want to verify

The remaining indexes contain values for the columns in the table.

Typically, this parameter is passed in from the HTML form, where you have previously called the *owa\_opt\_lock.store\_values* procedure to store the row values on hidden form elements.

## Return Value

TRUE if no other update has been performed; FALSE otherwise

## ***OWA\_PATTERN Package***

This package contains procedures and functions that you can use to perform string matching and string manipulation with regular expression functionality.

The OWA\_PATTERN package enables you to do sophisticated string manipulation using regular expressions. OWA\_PATTERN provides the following three operations:

- **MATCH** This determines whether a regular expression exists in a string. This is a function that returns TRUE or FALSE.
- **AMATCH** This is a more sophisticated variation on MATCH that lets you specify where in the string the match has to occur. This is a function that returns as an integer the end of the location in the string where the regular expression was found. If the regular expression is not found, it returns 0.
- **CHANGE** This lets you replace the portion of the string that matched the regular expression with a new string. CHANGE can be either a procedure or a function. If it's a function, it returns the number of times the regular expression was found and replaced.

Other operations can be used as well. The OWA\_PATTERN operations all use the following parameters:

- **line** This is the target to be examined for a match. Despite the name, it can be more than one line of text or can be a multi\_line PL/SQL table. (See *Oracle PL/SQL Tips and Techniques* (McGraw-Hill/Osborne) by Joseph C. Trezzo.)
- **pat** This is the regular expression in which the functions attempts to locate.
- **flags** This regular expression uses the special tokens.

## Regular Expressions

You specify a regular expression by creating the string you want to match interspersed with various wild card tokens and quantifiers. The wild card tokens all match something other than themselves, and the quantifiers modify the meaning of tokens or of literals by specifying such things as how often each is to be applied.

## Tokens

The wild card tokens that are supported are as follows:

^	Matches newline character or the beginning of the target.
\$	Matches newline character or the end of the target.
\n	Matches newline character.
.	Matches any character except newline character.
\t	Matches tab.
\d	Matches digits [0–9].
\D	Matches nondigits [not 0–9].
\w	Matches word characters (alphanumeric) [0–9, a–z, A–Z, or _].
\W	Matches nonword characters [not 0–9, a–z, A–Z, or _].
\s	Matches whitespace characters [blank, tab, or newline].

\S	Matches nonwhitespace characters [not blank, tab, or newline].
\b	Matches “word” boundaries (between \w and \W).
\x<HEX>	Matches the value in the current character set of the two hexadecimal digits.
\<OCT>	Matches the value in the current character set of the two or three octal digits.
\<char>	Matches <char> as long as <char> is not W, s, S, b, or x.
&	Applies only to CHANGE. This causes the string that matched the regular expression to be included in the string that replaces it. This differs from the other tokens in that it specifies how a target is changed rather than how it is matched. This is explained further under “owa_pattern.change”.

## Quantifiers

Any of the previously listed tokens except & can have their meaning extended by any of the following quantifiers. You can also apply these quantifiers to literals.

?	0 or 1 occurrence
*	0 or more occurrences
+	1 or more occurrences
{n}	Exactly <i>n</i> occurrences
(n,}	At least <i>n</i> occurrences
{n,m}	At least <i>n</i> , but not more than <i>m</i> , occurrences

## Flags

In addition to targets and regular expressions, the OWA\_PATTERN functions and procedures can use flags to affect how they are interpreted. The recognized flags are as follows:

i	This indicates a case-insensitive search.
g	This applies only to CHANGE. It indicates a global replace, which is to say,

	all portions of the target that match the regular expression are replaced.
--	--

## Datatypes

The following special datatype is used by OWA\_PATTERN:

pattern	A PL/SQL table of 4-byte varchar2varchar2 strings, indexed by BINARY INTEGER. This is an alternate way to store your regular expressions instead of using simple varchar2 strings. The advantage of this is that you can use a pattern as both an input and output parameter. Thus, you can pass the same regular expression to several subsequent OWA_PATTERN function calls, and the expression has to be parsed only once.
---------	---

Note: The following datatypes are used by OWA\_PATTERN but are part of the OWA\_TEXT package. For information on these datatypes, see the section “OWA\_TEXT” later in this appendix:

*owa\_text.vc\_array*, *owa\_text.multi\_line*, *owa\_text.int\_array*, or  
*owa\_text.row\_list*.

## Using MATCH, AMATCH, and CHANGE

Here is an example of MATCH:

```
MATCH ('BATMAN', 'Bat.*', i);
```

This is how the function is interpreted. BATMAN is the target where we are searching for the regular expression. Bat.\* is the regular expression we are attempting to find. The period (.) indicates any character other than a newline character, and the asterisk (\*) indicates 0 or more of such. Therefore, this regular expression specifies that a matching target consists of “Bat”, followed by any set of characters neither ending in nor including a newline character (which does not match the period). The “i” at the end is a flag indicating that case is to be ignored in the search. This would return TRUE, if a match is found.

Tip: If multiple overlapping strings can match the regular expression, OWA\_PATTERN takes the longest match.

***owa\_pattern.match (version 1)***

Enables programmers to search a string for a pattern using regular expressions.

**Syntax**

```
owa_pattern.match(line, pat, flags)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
line	in	varchar2	No	Line to be searched
pat	in	varchar2	No	Pattern to be searched for in line
flags	in	varchar2	Yes, default is NULL	Flags as defined previously

**Generates**

Boolean indicating whether match was found

***owa\_pattern.match (version 2)***

Enables programmers to search a string for a pattern using regular expressions.

**Syntax**

```
owa_pattern.match(line, pat, flags)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
line	in	varchar2	No	Line to be searched
pat	in	varchar2	No	Pattern to be searched for

				in line
flags	in	varchar2	Yes, default is NULL	Flags as defined previously

**Generates**

Boolean indicating whether match was found

***owa\_pattern.match (version 3)***

Enables programmers to search a string for a pattern using regular expressions.

**Syntax**

```
owa_pattern.match(line, pat, backrefs, flags)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
line	in	varchar2	No	Line to be searched.
pat	in	varchar2	No	Pattern to be searched for in lines.
backrefs	out	owa_text.vc_ arr	No	This is a PL/SQL table that holds each string in the target that was matched by a sequence of tokens in the regular expression.
flags	in	varchar2	Yes, default is NULL	Flags as defined previously.

**Generates**

Boolean indicating whether match was found

***owa\_pattern.match (version 4)***

Enables programmers to search a string for a pattern using regular expressions.

**Syntax**

```
owa_pattern.match(line, pat, backrefs, flags)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
line	in	varchar2	No	Line to be searched.
pat	in	varchar2	No	Pattern to be searched for in lines.
backrefs	out	owa_text.vc_ arr	No	This is a PL/SQL table that holds each string in the target that was matched by a sequence of tokens in the regular expression.
flags	in	varchar2	Yes, default is NULL	Flags as defined previously.

**Generates**

Boolean indicating whether match was found

***owa\_pattern.match (version 5)***

Enables programmers to search a string for a pattern using regular expressions.

**Syntax**

```
owa_pattern.match(mline, pat, rlist, flags)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
mline	in	owa_text.multi_line	No	Array of lines to search.
pat	in	varchar2	No	Pattern to search for in lines.
rlist	out	owa_text.row_list	No	This is a PL/SQL table that holds each string in the target that was matched by a sequence of tokens in the regular expression.
flags	in	varchar2	Yes, default is NULL	Flags as defined previously.

**Generates**

Boolean indicating whether match was found

***owa\_pattern.match (version 6)***

Enables programmers to search a string for a pattern using regular expressions.

**Syntax**

```
owa_pattern.match(mline, pat, rlist, flags)
```

**Parameters**

Parameter	In, Out, or In	Datatype	Optional?	Description
-----------	----------------	----------	-----------	-------------

	<b>Out</b>			
mline	in	owa_text.multi_line	No	Array of lines to search.
pat	in	varchar2	No	Pattern to search for in lines.
rlist	out	owa_text.row_list	No	This is a PL/SQL table that holds each string in the target that was matched by a sequence of tokens in the regular expression.
flags	in	varchar2	Yes, default is NULL	Flags as defined previously.

**Generates**

Boolean indicating whether match was found

***owa\_pattern.amatch (version 1)***

Enables programmers to search a string for a pattern using regular expressions.

**Syntax**

```
owa_pattern.amatch(line, from_loc, pat, flags)
```

**Parameters**

<b>Parameter</b>	<b>In, Out, or In Out</b>	<b>Datatype</b>	<b>Optional?</b>	<b>Description</b>
line	in	varchar2	No	Line to search.
from_loc	in	integer	No	This indicates how many

				characters from the beginning of the target the search should commence.
Pat	in	varchar2	No	Pattern to search for in lines.
flags	in	varchar2	Yes, default is NULL	Flags as described previously.

### Generates

Location (in number of characters from the beginning) to the end of the match, 0 if there are none

### ***owa\_pattern.amatch (version 2)***

Enables programmers to search a string for a pattern using regular expressions.

### Syntax

```
owa_pattern.amatch(line, from_loc, pat, flags)
```

### Parameters

Parameter	In, Out, or In Out	Datatype	Optional?	Description
line	in	varchar2	No	Line to search.
from_loc	in	integer	No	This indicates how many characters from the beginning of the target the search should commence.
pat	in	varchar2	No	Pattern to search for in

				lines.
flags	in	varchar2	Yes, default is NULL	Flags as described previously.

### Generates

Location (in number of characters from the beginning) to the end of the match, 0 if there are none

### ***owa\_pattern.amatch (version 3)***

Enables programmers to search a string for a pattern using regular expressions.

### Syntax

```
owa_pattern.amatch(line, from_loc, pat, backrefs, flags)
```

### Parameters

Parameter	In, Out, or In Out	Datatype	Optional?	Description
line	in	varchar2	No	Line to search.
from_loc	in	integer	No	This indicates how many characters from the beginning of the target the search should commence.
pat	in	varchar2	No	Pattern to search for in lines.
backrefs	out	owa_text.vc_a rr	No	This is a PL/SQL table that holds each string in the target that was matched by a sequence of tokens in the regular

				expression.
flags	in	varchar2	Yes, default is NULL	Flags as described previously.

### Generates

Location (in number of characters from the beginning) to the end of the match, 0 if there are none

### ***owa\_pattern.amatch (version 4)***

Enables programmers to search a string for a pattern using regular expressions.

### Syntax

```
owa_pattern.amatch(line, from_loc, pat, backrefs, flags)
```

### Parameters

Parameter	In, Out, or In Out	Datatype	Optional?	Description
line	in	varchar2	No	Line to search.
from_loc	in	integer	No	This indicates how many characters from the beginning of the target the search should commence.
pat	in	varchar2	No	Pattern to search for in lines.
backrefs	out	owa_text.vc_a rr	no	This is a PL/SQL table that holds each string in the target that was matched by a sequence of tokens in the

				regular expression.
flags	in	varchar2	Yes, default is NULL	Flags as described previously.

**Generates**

Location (in number of characters from the beginning) to the end of the match, 0 if there are none

***owa\_pattern.change (version 1)***

This version is a function. It enables programmers to search a string for a pattern and replace it.

**Syntax**

```
owa_pattern.change(line, from_str, to_str, flags)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
line	in out	varchar2	No	Line to perform search and replace operation on
from_str	in	varchar2	No	String to search for
to_str	in	varchar2	No	String to replace from_str with
flags	in	varchar2	Yes, default is NULL	Flags as defined previously

**Generates**

Revises line parameter. Function outputs number of substitutions made.

***owa\_pattern.change (version 2)***

This version is a procedure. It enables programmers to search a string for a pattern and replace it.

**Syntax**

```
owa_pattern.change(line, from_str, to_str, flags)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
line	in out	varchar2	No	Line to perform search and replace operation on
from_str	in	varchar2	No	String to search for
to_str	in	varchar2	No	String to replace from_str with
flags	in	varchar2	Yes, default is NULL	Flags as defined previously

**Generates**

Revises line parameter

***owa\_pattern.change (version 3)***

This version is a function. It enables programmers to search a multi\_line for a pattern and replace it.

**Syntax**

```
owa_pattern.change(mline, from_str, to_str, backrefs, flags)
```

**Parameters**

<b>Parameter</b>	<b>In, Out, or In Out</b>	<b>Datatype</b>	<b>Optional?</b>	<b>Description</b>
<code>mline</code>	in out	<code>varchar2</code>	No	Lines to perform search and replace operation on
<code>from_str</code>	in	<code>varchar2</code>	No	String to search for
<code>to_str</code>	in	<code>varchar2</code>	No	String to replace <code>from_str</code> with
<code>flags</code>	in	<code>varchar2</code>	Yes, default is NULL	Flags as defined previously

**Generates**

Revises *mline* parameter. Function outputs number of substitutions made.

***owa\_pattern.change (version 4)***

This version is a procedure. It enables programmers to search a `multi_line` for a pattern and replace it.

**Syntax**

```
owa_pattern.change(mline, from_str, to_str, flags)
```

**Parameters**

<b>Parameter</b>	<b>In, Out, or In Out</b>	<b>Datatype</b>	<b>Optional?</b>	<b>Description</b>
<code>mline</code>	in out	<code>varchar2</code>	No	Lines to perform search and replace operation on
<code>from_str</code>	in	<code>varchar2</code>	No	String to search for
<code>to_str</code>	in	<code>varchar2</code>	No	String to replace <code>from_str</code>

				with
flags	in	varchar2	Yes, default is NULL	Flags as defined previously

**Generates**

Revises *mline* parameter

***owa\_pattern.getpat***

This converts a varchar2 string into the special datatype pattern. This datatype is explained in “Datatypes” in the section “OWA\_PATTERN.”

**Syntax**

```
owa_pattern.getpat (arg, pat)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
arg	in	varchar2	No	Argument to convert
Pat	in out	owa_util.pa ttern	No	Pattern returned from conversion

**Generates**

Pattern

**OWA\_SEC Package**

This section describes the functions, procedures, and datatypes in the OWA\_SEC package in the PL/SQL Web Toolkit. Parameters that have default values are optional. This package contains functions and procedures used by the cartridge for authenticating requests.

***owa\_sec.get\_client\_hostname Function***

This function returns the hostname of the client.

**Syntax**

```
owa_sec.get_client_hostname return varchar2;
```

**Parameters**

None

**Return Value**

The hostname

***owa\_sec.get\_client\_ip Function***

This function returns the IP address of the client.

**Syntax**

```
owa_sec.get_client_ip return owa_util.ip_address;
```

**Parameters**

None

**Return Value**

This function returns the IP address of the client. The `owa_util.ip_address` datatype is a PL/SQL table in which the first four elements contain the four numbers of the IP address. For example, if the IP address is 123.45.67.89 and the variable `ipaddr` is of the `owa_util.ip_address` datatype, the variable would contain the following values:

```
ipaddr(1) = 123  
ipaddr(2) = 45  
ipaddr(3) = 67  
ipaddr(4) = 89
```

***owa\_sec.get\_password Function***

This function returns the password that the user used to log in.

**Syntax**

```
owa_sec.get_password return varchar2;
```

**Parameters**

None

**Return Value**

The password

***owa\_sec.get\_user\_id Function***

This function returns the username that the user used to log in.

**Syntax**

```
owa_sec.get_user_id return varchar2;
```

**Parameters**

None

**Return Value**

The username

***owa\_sec.set\_authorization Procedure***

This procedure sets the authorization scheme for a PL/SQL Agent. Setting the scheme parameter to GLOBAL or PER\_PACKAGE enables you to define your own authentication routine. This procedure is called in the initialization portion of the owa\_init package.

**Syntax**

```
owa_sec.set_authorization(scheme in integer);
```

**Parameters**

Sets the authorization scheme, which is one of the following:

- **OWA\_SEC.NO\_CHECK** (default) Specifies that the PL/SQL application is not to do any custom authentication.
- **OWA\_SEC.GLOBAL** Specifies that the *owa\_init.authorize* function is to be used to authorize the user. You have to define this function in the OWA\_INIT package.
- **OWA\_SEC.PER\_PACKAGE** Specifies that the *package.authorize* function or the anonymous authorize function is to be used to authorize the user. You have to define this function. If the request is for a procedure defined in a package, the *package.authorize* function is called. If the request is for a procedure that is not in a package, the anonymous authorize function is called.

### ***owa\_sec.set\_protection\_realm Procedure***

This procedure sets the realm of the page that is returned to the user. The user must enter a username and login that already exists in the realm for authorization to succeed. Realms are established for a specific listener under the security section of its configuration options and can be established as basic or digest authorization realms.

#### **Syntax**

```
owa_sec.set_protection_realm(realm in varchar2);
```

#### **Parameters**

- **Realm** The realm in which the page should belong. This string is displayed to the user.

### **OWA\_TEXT Package**

The OWA\_TEXT package is chiefly used by OWA\_PATTERN, but the functions are externalized so that you can use them directly if desired. This package contains procedures, functions, and datatypes used by OWA\_PATTERN for manipulating large data strings. They are externalized so you can use them directly.

#### ***Datatypes***

vc_array	A PL/SQL table of 32KB varchar2 strings, indexed by BINARY
----------	--

	INTEGER. This is a component of owa_text.multi_line.
multi_line	A record with three fields.
int_array	A PL/SQL table of INTEGER indexed by BINARY INTEGER. This is a component of owa_text.row_list.
row_list	A record with fields.

### ***owa\_text.stream2multi***

Converts a long string to a multi\_line.

#### **Syntax**

```
owa_text.stream2multi(stream, mline)
```

#### **Parameters**

<b>Parameter</b>	<b>In, Out, or In Out</b>	<b>Datatype</b>	<b>Optional?</b>	<b>Description</b>
stream	in	varchar2	No	Stream of text in a long field
mline	out	owa_util. multi_line	No	Stream converted to a multi_line variable

#### **Generates**

```
multi_line
```

### ***owa\_text.add2multi***

Adds more content to a multi\_line. The continue parameter specifies whether to begin appending within the previous final chunk (assuming it is less than 32KB) or to start a new chunk.

#### **Syntax**

```
owa_text.add2multi(stream, mline, continue)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
stream	in	varchar2	No	Stream of text in a long field.
mline	out	owa_util. multi_line	No	Output from converted long field into a multi_line field.
continue	in	Boolean	Yes, default is TRUE	TRUE or FALSE. If TRUE, append to previous value.

**Generates**

```
|multi_line
```

***owa\_text.new\_row\_list***

Can be a procedure or a function. If a function it takes no parameters and outputs a new, initially empty, row\_list. If a procedure, it places into the rlist parameter a new, initially empty row\_list.

**Syntax**

```
|owa_text.new_row_list(rlist)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
rlist	in	row_list	No	Row list to place into a new, empty row_list

**Generates**

Actual output generated by htp.print

***owa\_text.print\_multi***

Uses `http.print` to print the `multi_line`.

**Syntax**

```
owa_text.print_multi(mline)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
<code>mline</code>	<code>in</code>	<code>multi_line</code>	No	<code>Multi_line</code> variable to print

**Generates**

Actual output generated by `http.print`

***owa\_text.print\_row\_list***

Uses `http.print` to print the `row_list`.

**Syntax**

```
owa_text.print_row_list(rlist)
```

**Parameters**

Parameter	In, Out, or In Out	Datatype	Optional?	Description
<code>rlist</code>	<code>in</code>	<code>row_list</code>	No	<code>row_list</code> to print.

**Generates**

Actual output generated by `http.print`

**OWA\_UTIL Package**

This section describes the functions, procedures, and datatypes in the `OWA_UTIL` package in the PL/SQL Web Toolkit. Parameters that have default values

are optional. This package contains utility procedures and functions. Each function, procedure, and datatype is marked as fitting into the following areas:

- **HTML Utilities** The purposes of these utilities range from printing a signature tag on HTML pages to retrieving the values of CGI environment variables and performing URL redirects.
- **Dynamic SQL Utilities** These utilities enable you to produce web pages with dynamically generated SQL code.
- **Date Utilities** These utilities make it easier to properly handle dates, which are simple strings in HTML, but are properly treated as a datatype by the Oracle RDBMS.
- **Miscellaneous** These do not fit into one of the previously listed categories.

### ***owa\_util.bind\_variables Function (Dynamic SQL Utility)***

This function prepares a SQL query by binding variables to it and stores the output in an opened cursor. You normally use this function as a parameter to a procedure to which you desire to send a dynamically generated query. You can specify up to 25 bind variables.

#### **Syntax**

```
owa_util.bind_variables(  
    theQuery          in    varchar2 DEFAULT NULL  
    bv1Name           in    varchar2 DEFAULT NULL  
    bv1Value          in    varchar2 DEFAULT NULL  
    bv2Name           in    varchar2 DEFAULT NULL  
    bv2Value          in    varchar2 DEFAULT NULL  
    bv3Name           in    varchar2 DEFAULT NULL  
    bv3Value          in    varchar2 DEFAULT NULL  
    ...  
    bv25Name          in    varchar2 DEFAULT NULL  
    bv25Value         in    varchar2 DEFAULT NULL) return integer;
```

#### **Parameters**

- **theQuery** The SQL query statement. This must be a SELECT statement.

- **bv1Name** The name of the variable.
- **bv2Value** The value of the variable.

### **Return Value**

An integer identifying the opened cursor

### ***owa\_util.calendarprint Procedure (Date Utility)***

This procedure creates a calendar in HTML. Each date in the calendar can contain any number of hypertext links. To achieve this effect, design your query as follows:

- The first column should be DATE. This is used to correlate the information produced by the query with the calendar output automatically generated by the procedure.

**Note:** The query output must be sorted on this column using ORDER BY.

- The second column contains the text, if any, that you want printed for that date.
- The third column contains the destination for automatically generated links. Each item in the second column becomes a hypertext link to the destination given in this column. If this column is omitted, the items in the second column are simple text, not links.

This procedure has two versions. Version 1 uses a hard-coded query stored in a varchar2 string. Version 2 uses a dynamic query prepared with the *owa\_util.bind\_variables* function.

### ***owa\_util.choose\_date Procedure (Date Utility)***

This procedure generates three HTML form elements that allow the user to select the day, the month, and the year.

The parameter in the procedure that receives the data from these elements should be an *owa\_util.dateType* datatype. You can use the *owa\_util.todate* function to convert the *owa\_util.dateType* datatype value to the standard Oracle7 DATE datatype.

## Syntax

```
owa_util.choose_date(  
    p_name in varchar2,  
    p_date in date DEFAULT SYSDATE);
```

## Parameters

- **p\_name** The name of the form elements
- **p\_date** The initial date that is selected when the HTML page is displayed

## Generates

```
<SELECT NAME="p_name" SIZE="1">  
<OPTION value="01">1  
<OPTION value="02">2  
<OPTION value="03">3  
<OPTION value="04">4  
<OPTION value="05">5  
<OPTION value="06">6  
<OPTION value="07">7  
<OPTION value="08">8  
<OPTION value="09">9  
<OPTION value="10">10  
<OPTION value="11">11  
<OPTION value="12">12  
<OPTION value="13">13  
<OPTION value="14">14  
<OPTION value="15">15  
<OPTION value="16">16  
<OPTION value="17">17  
<OPTION value="18">18  
<OPTION value="19">19  
<OPTION value="20">20  
<OPTION value="21">21  
<OPTION value="22">22  
<OPTION value="23">23  
<OPTION SELECTED value="24">24  
<OPTION value="25">25  
<OPTION value="26">26
```

```
<OPTION value="27">27
<OPTION value="28">28
<OPTION value="29">29
<OPTION value="30">30
<OPTION value="31">31
</SELECT>
--
<SELECT NAME="p_name" SIZE="1">
<OPTION value="01">JAN
<OPTION SELECTED value="02">FEB
<OPTION value="03">MAR
<OPTION value="04">APR
<OPTION value="05">MAY
<OPTION value="06">JUN
<OPTION value="07">JUL
<OPTION value="08">AUG
<OPTION value="09">SEP
<OPTION value="10">OCT
<OPTION value="11">NOV
<OPTION value="12">DEC
</SELECT>
--
<SELECT NAME="p_name" SIZE="1">
<OPTION value="1992">1992
<OPTION value="1993">1993
<OPTION value="1994">1994
<OPTION value="1995">1995
<OPTION value="1996">1996
<OPTION SELECTED value="1997">1997
<OPTION value="1998">1998
<OPTION value="1999">1999
<OPTION value="2000">2000
<OPTION value="2001">2001
<OPTION value="2002">2002
</SELECT>
```

### ***owa\_util.dateType Datatype (Date Datatype)***

This datatype holds date information. It is defined as:

```
type dateType is table of varchar2(10) index by binary_integer
```

The *owa\_util.to\_date* function converts an item of this type to the type DATE, which is understood and properly handled as data by the database. The procedure *owa\_util.choose\_date* procedure enables the user to select the desired date.

### ***owa\_util.get\_cgi\_env* Function (Miscellaneous)**

This function returns the value of the specified CGI environment variable. Although the WRB is not operated through CGI, many WRB cartridges, including the PL/SQL cartridge, can make use of CGI environment variables.

#### **Syntax**

```
owa_util.get_cgi_env(param_name in varchar2) return varchar2;
```

#### **Parameters**

- **param\_name** The name of the CGI environment variable. It is case-insensitive.

For example, the following code would retrieve the IP address of the remote node (the client's PC):

```
remote_ip := owa_util.get_cgi_env('REMOTE_ADDR');
```

The following parameters may be obtained using this function, and additional parameters may be available depending upon the operating system:

- **SERVER\_SOFTWARE** The name of the server handling the request (i.e., the web server software in use)—for example, Oracle\_Web\_listener2.0/1.20in2 or Oracle\_Web\_Listener3.0/3.0.0.0.
- **SERVER\_NAME** The domain name of the computer that is running the server software (i.e., the domain name of the web server)—for example, tuscil\_db1 or www.tusc.com.
- **GATEWAY\_INTERFACE** The name and version of the gateway interface that is being used. The gateway interface name and version are separated by a forward slash with no spaces—for example, CGI/1.1.

- **REMOTE\_HOST** The name of the remote computer that made the request—for example, laptop2, tusc\_bdb, or 192.103.202.40.
- **REMOTE\_ADDR** The IP address of the client that made the request—for example, 209.108.212.250.
- **HTTP\_ACCEPT** The contents of the Accept: header line sent by the client. This line contains a list of MIME types that the client can handle, separated by commas *without spaces*—for example, image/gif,image/x-xbitmap,image/jpeg,image/pjpeg,\*/\*.
- **HTTP\_USER\_AGENT** The name of the client program making the request—for example, Mozilla/3.01 (WinNT; I). This tells you what browser and operating system is being used by the user.
- **SERVER\_PROTOCOL** The name of the protocol that the server is using and the version of the protocol. The protocol name and version are separated by a forward slash and no spaces—for example, HTTP/1.0.
- **SERVER\_PORT** The TCP port number on which the server that invoked the CGI application is operating—for example, 80 (this is the default HTTP port number).
- **SCRIPT\_NAME** The name of the script that was involved. For Oracle Application Server, this is the PL/SQL agent name, such as /bdb/owa.
- **PATH\_INFO** Extra path information as it was passed to the server in the query URL. For Oracle Application Server, this is the name of the (packaged) procedure that was called—for example, /owa\_util.print\_cgi\_env.
- **PATH\_TRANSLATED** Extra path information in the query URL, translated into a final, usable form (for the WebServer) —for example, c:\orant\ows20\bin\owa\_util.print\_cgi\_env.

### ***owa\_util.get\_owa\_service\_path Function (Miscellaneous)***

This function returns the full virtual path of the PL/SQL cartridge that is handling the request.

Note: See the `owa_util.get_cgi_env` code example (i.e., `owa_util_example`) in the preceding section.

**Syntax**

```
owa_util.get_owa_service_path return varchar2;
```

**Parameters**

None

**Return Value**

A virtual path of the PL/SQL cartridge that is handling the request

***owa\_util.get\_procedure Function (Miscellaneous)***

This function returns the name of the procedure that is being invoked by the PL/SQL Agent.

Note: See the `owa_util.get_cgi_env` code example (i.e., the `owa_util_example`) in the section “`owa_util.get_cgi_env` Function (Miscellaneous)”.

**Syntax**

```
owa_util.get_procedure return varchar2;
```

**Parameters**

None

**Return Value**

The name of a procedure, including the package name if the procedure is defined in a package

***owa\_util.http\_header\_close Procedure (Miscellaneous)***

This procedure generates a newline character to close the HTTP header. Use this procedure if you have not explicitly closed the header by using the `bclose_header` parameter in calls such as the `owa_util.mime_header` procedure, the `owa_util.redirect_url`

procedure, or the `owa_util.status_line` procedure. The HTTP header must be closed before any `htp.print` or `htp.prn` calls.

Note: See the code examples in the section “OWA\_COOKIE Package”.

### Syntax

```
owa_util.http_header_close;
```

### Parameters

None

### Generates

A newline character, which closes the HTTP header

### ***owa\_util.ident\_arr Datatype (Miscellaneous)***

This datatype is defined as:

```
type ident_arr is table of varchar2(30) index by binary_integer
```

This datatype is used for form pages that have multiple parameters with the same name—for example, checkboxes or drop-down list boxes (select).

### ***owa\_util.ip\_address Datatype (Miscellaneous)***

This datatype is defined as:

```
type ip_address is table of integer index by binary_integer
```

This datatype is used by the `owa_sec.get_client_ip` function.

### ***owa\_util.listprint Procedure (HTML Utility)***

This procedure generates an HTML selection list form element from the output of a SQL query. The columns in the output of the query are handled in the following manner:

- The first column specifies the values that are sent back. These values are used for the VALUE attribute of the OPTION tag.
- The second column specifies the values that the user sees.

- The third column specifies whether or not the row is marked as **SELECTED** in the **OPTION** tag. If the value is not **NULL**, the row is selected.

There are two versions of this procedure. The first version contains a hard-coded SQL query, and the second version uses a dynamic query prepared with the *owa\_util.bind\_variables* function.

### Syntax

```
owa_util.listprint (
    p_theQuery      in    varchar2
    p_cname         in    varchar2
    p_nsize         in    number
    p_multiple      in    boolean    DEFAULT FALSE);

owa_util.listprint (
    p_theCursor     in    integer
    p_cname         in    varchar2
    p_nsize         in    number
    p_multiple      in    boolean    DEFAULT FALSE);
```

### Parameters

- **p\_theQuery** The SQL query.
- **p\_theCursor** The cursor ID—this can be the return value from the *owa\_util.bind\_variables* function.
- **p\_cname** The name of the HTML form element.
- **p\_nsize** The size of the form element (which controls how many items the user can see without scrolling).
- **p\_multiple** Indicates whether multiple selection is permitted.

### Generates

```
<SELECT NAME="p_cname" SIZE="p_nsize">
<OPTION SELECTED
value='value_from_the_first_column'>value_from_the_second_column
```

```
<OPTION SELECTED  
value='value_from_the_first_column'>value_from_the_second_column  
  
...  
  
</SELECT>
```

### ***owa\_util.mime\_header Procedure (Miscellaneous)***

This procedure changes the default MIME header that the PL/SQL Agent returns. This procedure must come before any `http.print` or `http.prn` calls in order to direct the PL/SQL Agent not to use the default.

Note: See the code examples in the section “OWA\_COOKIE Package”.

#### **Syntax**

```
owa_util.mime_header(  
    ccontent_type      in    varchar2 DEFAULT `text/html`,  
    bclose_header     in    boolean DEFAULT TRUE);
```

#### **Parameters**

- **ccontent\_type** The MIME type to generate.
- **bclose\_header** Indicates whether or not to close the HTTP header. If TRUE, two new lines are sent, which closes the HTTP header. Otherwise, one new line is sent, and the HTTP header is still open.

#### **Generates**

```
Content-type: <ccontent_type>\n\n
```

### ***owa\_util.print\_cgi\_env Procedure (Miscellaneous)***

This procedure generates all the CGI environment variables and their values made available by the PL/SQL Agent to the PL/SQL procedure.

Note: See the `owa_util.get_cgi_env` code example (i.e., the `owa_util_example`) in the section “`owa_util.get_cgi_env` Function (Miscellaneous)”.

**Syntax**

```
owa_util.print_cgi_env;
```

**Parameters**

None

**Generates**

A list in the following format:

```
cgi_env_var_name = value\n
```

***owa\_util.redirect\_url Procedure (Miscellaneous)***

This procedure specifies that the Application Server is to visit the specified URL. The URL may specify either a web page to return or a program to execute. This procedure must come before any `htp.print` or `htp.prn` calls in order to tell the PL/SQL Agent to do the redirect.

**Syntax**

```
owa_util.redirect_url(  
    curl          in    varchar2  
    bclose_header in    boolean DEFAULT TRUE);
```

**Parameters**

- **curl** The URL to visit.
- **bclose\_header** Indicates whether or not to close the HTTP header. If TRUE, two new lines are sent, which closes the HTTP header. Otherwise, one new line is sent, and the HTTP header is still open.

**Generates**

```
Location: <curl>\n\n
```

***owa\_util.showpage Procedure (Miscellaneous)***

This procedure prints out the HTML output of a procedure in SQL\*Plus, SQL\*DBA, or Oracle Server Manager. The procedure must use the HTP or HTF

packages to generate the HTML page, and this procedure must be issued after the procedure has been called and before any other HTP or HTF subprograms are directly or indirectly called. This method is useful for generating pages filled with static data.

Tip: This procedure uses `dbms_output` and is limited to 255 characters per line and an overall buffer size of 1,000,000 bytes.

### Syntax

```
owa_util.showpage;
```

### Parameters

None

### Generates

The output of HTP procedure is displayed in SQL\*Plus, SQL\*DBA, or Oracle Server Manager. For example:

```
SQL> set serveroutput on
SQL> spool gretzky.html
SQL> execute hockey.pass('Gretzky')
SQL> execute owa_util.showpage
SQL> exit
```

This would generate an HTML page that could be accessed from web browsers.

Tip: This procedure is also useful if you don't spool the information to an HTML file, but only to the screen. By reading the HTML, you can quickly diagnose a problem with the data. From SQL\*Plus you may also find that it is easier to debug the problem because the error message may be more descriptive too.

### ***owa\_util.showsouce Procedure (Miscellaneous)***

This procedure prints the source of the specified procedure, function, or package. If a procedure or function is specified that belongs to a package, then the entire package is displayed.

Note: See the `owa_util.get_cgi_env` code example (i.e., the `owa_util_example`) in the section “`owa_util.get_cgi_env` Function (Miscellaneous)”.

### Syntax

```
owa_util.showsource (cname in varchar2);
```

### Parameters

- **cname** Name of the procedure or function

### Generates

The source code of the specified function, procedure, or package

### ***owa\_util.signature Procedure (HTML Utility)***

This procedure generates an HTML line followed by a signature line on the HTML document. If a parameter is specified, the procedure also generates a hypertext link to view the PL/SQL source for that procedure. The link calls the `owa_util.showsource` procedure.

Note: See the `owa_util.get_cgi_env` code example (i.e., the `owa_util_example`) in the section “`owa_util.get_cgi_env` Function (Miscellaneous)”.

### Syntax

```
owa_util.signature;  
owa_util.signature (cname in varchar2);
```

### Parameters

- **cname** The function or procedure whose source you want to show

### Generates

Without a parameter, the procedure generates a line that looks like the following:

```
This page was produced by the PL/SQL Agent on August 9, 1995 09:30
```

With a parameter, the procedure generates a signature line in the HTML document that might look like the following:

This page was produced by the PL/SQL Agent on 6/14/95 09:30

[View PL/SQL Source](#)

### ***owa\_util.status\_line Procedure (HTML Utility)***

This procedure sends a standard HTTP status code to the client. This procedure must come before any `htp.print` or `htp.prn` calls so that the status code is returned as part of the header, rather than as “content data.”

Note: See the `owa_util.get_cgi_env` code example (i.e., the `owa_util_example`) in the section “`owa_util.get_cgi_env` Function (Miscellaneous)”.

#### **Syntax**

```
owa_util.status_line(  
    nstatus          in    integer,  
    creason          in    varchar2 DEFAULT NULL  
    bclose_header   in    boolean DEFAULT TRUE);
```

#### **Parameters**

- **nstatus** The status code.
- **creason** The string for the status code.
- **bclose\_header** Indicates whether or not to close the HTTP header. If TRUE, two new lines are sent, which closes the HTTP header. Otherwise, one new line is sent, and the HTTP header is still open.

#### **Generates**

```
Status: <nstatus> <creason>\n\n
```

### ***owa\_util.tablePrint Function (Dynamic SQL Utility)***

This function generates either preformatted or HTML tables (depending on the capabilities of the user’s browser) from database tables. Note that RAW columns are supported, but LONG RAW columns are not. References to LONG RAW columns print “Not Printable”. In this function, *attributes* is the second rather than the last parameter.

**Syntax**

```

owa_util.tablePrint (
    ctable          in    varchar2
    cattributes     in    varchar2    DEFAULT NULL
    ntable_type     in    integer     DEFAULT HTML_TABLE
    ccolumns        in    varchar2    DEFAULT `*`
    cclauses        in    varchar2    DEFAULT NULL
    ccol_aliases    in    varchar2    DEFAULT NULL
    nrow_min        in    number      DEFAULT 0
    nrow_max        in    number      DEFAULT NULL )
return boolean;

```

**Parameters**

- **ctable** The database table.
- **cattributes** Other attributes to be included, as is, in the tag.
- **ntable\_type** How to generate the table. Specify “HTML\_TABLE” to generate the table using <TABLE> tags or “PRE\_TABLE” to generate the table using the <PRE> tags.
- **ccolumns** A comma-delimited list of columns from ctable to include in the generated table.
- **cclauses** WHERE or ORDER BY clauses, which let you specify which rows to retrieve from the database table, and how to order them.
- **ccol\_aliases** A comma-delimited list of headings for the generated table.
- **nrow\_min** The first row, of those retrieved, to display.
- **nrow\_max** The last row, of those retrieved, to display.

**Generates**

A preformatted or HTML table

**Return Value**

TRUE if there are more rows beyond the nrow\_max requested, FALSE otherwise

### ***owa\_util.todate Function (Date Utility)***

This function converts the `owa_util.dateType` datatype to the standard Oracle database DATE type.

#### **Syntax**

```
owa_util.todate(p_dateArray in dateType) return date;
```

#### **Parameters**

- **p\_dateArray** The value to convert

#### **Generates**

A standard DATE

### ***owa\_util.who\_called\_me Procedure (Miscellaneous)***

This procedure returns information (in the form of output parameters) about the PL/SQL code unit that invoked it.

#### **Syntax**

```
owa_util.who_called_me (  
    owner          out   varchar2  
    name           out   varchar2  
    lineno         out   number  
    caller_t       out   varchar2);
```

#### **Parameters**

- **owner** The owner of the program unit.
- **name** The name of the program unit. This is the name of the package, if the calling program unit is wrapped in a package, and the name of the procedure or function if the calling program unit is a standalone procedure or function. If the calling program unit is part of an anonymous block, this is NULL.
- **lineno** The line number within the program unit where the call was made.

- **caller\_t** The type of program unit that made the call. The possibilities are package body, anonymous block, procedure, and function. Procedure and function are used only for standalone procedures and functions.