TR/WD-tables-960123

WD-tables-960123

# The HTML3 Table Model

### W3C Working Draft 23-Jan-96

This version:
http://www.w3.org/pub/WWW/TR/WD-tables-960123.html
Latest version:
http://www.w3.org/pub/WWW/TR/WD-tables.html
Editor:
Dave Raggett <dsr@w3.org>

---

## Status of this document

This is a W3C Working Draft for review by W3C members and other interested parties. It is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". A list of current W3C working drafts can be found at: http://www.w3.org/pub/WWW/TR

**Note:** since working drafts are subject to frequent change, you are advised to reference the above URL, rather than the URLs for working drafts themselves.

## Abstract

The HyperText Markup Language (HTML) is a simple markup language used to create hypertext documents that are portable from one platform to another. HTML documents are SGML documents with generic semantics that are appropriate for representing information from a wide range of applications. This specification extends HTML to support a wide variety of tables. The model is designed to work well with associated style sheets, but does not require them. It also supports rendering to braille, or speech, and exchange of tabular data with databases and spreadsheets. The HTML table model embodies certain aspects of the CALS table model, e.g. the ability to group table rows into thead, tbody and tfoot sections, plus the ability to specify cell alignment compactly for sets of cells according to the context.

---

## Contents



## Recent Changes

This specification extends HTML to support tables. The table model has grown out of early work on

HTML+ and the initial draft of HTML3. The earlier model has been been extended in response to requests from information providers for improved control over the presentation of tabular information:

- alignment on designated characters such as "." and ":"
  e.g. aligning a column of numbers on the decimal point
- more flexibility in specifying table frames and rules
- incremental display for large tables as data is received
- the ability to support scrollable tables with fixed headers plus better support for breaking tables across pages for printing
- optional column based defaults for alignment properties

In addition, a major goal has been to provide backwards compatibility with the widely deployed Netscape implementation of tables. A subsidiary goal has been to simplify importing tables conforming to the SGML CALS model. The latest draft makes the `ALIGN` attribute compatible with the latest Netscape and Microsoft browsers. Some clarifications have been made to the role of the `DIR` attribute and recommended behaviour when absolute and relative column widths are mixed.

A new element `COLGROUP` has been introduced to allow sets of columns be grouped with different width and alignment properties specified by one or more `COL` elements. The semantics of COLGROUP have been clarified over previous drafts, and `RULES=BASIC` replaced by `RULES=GROUPS`.

The `STYLE` attribute is included as a means for extending the properties associated with edges and interiors of groups of cells. For instance, the line style: dotted, double, thin/thick etc; the colour/pattern fill for the interior; cell margins and font info. This will be the subject for a companion specification on style sheets.

The `FRAME` and `RULES` attributes have been modified to avoid SGML name clashes with each other, and to avoid clashes with the `ALIGN` and `VALIGN` attributes. These changes were additionally motivated by the desire to avoid future problems if this specification is extended to allow `FRAME` and `RULES` attributes with other table elements.

---

# A Brief Introduction to HTML Tables

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 include attributes assisting rendering to speech and braille, or for exporting table data into databases. The model provides limited support for control over appearence, for example horizontal and vertical alignment of cell contents, border styles and cell margins. You can further affect this by grouping rows and columns together. Tables can contain a wide range of content, such as headers, lists, paragraphs, forms, figures, preformatted text and even nested tables.

## Example

```
<TABLE BORDER>
  <CAPTION>A test table with merged cells</CAPTION>
  <TR><TH ROWSPAN=2><TH COLSPAN=2>Average
      <TH ROWSPAN=2>other<BR>category<TH>Misc
  <TR><TH>height<TH>weight
  <TR><TH ALIGN=LEFT>males<TD>1.9<TD>0.003
  <TR><TH ALIGN=LEFT ROWSPAN=2>females<TD>1.7<TD>0.002
</TABLE>
```

On a dumb terminal, this would be rendered something like:

```
          A test table with merged cells
/-----------------------------------------------\
|          |      Average      |  other   | Misc |
|          |-------------------| category |------|
|          |  height |  weight |          |      |
|----------|---------|---------|----------|------|
| males    | 1.9     | 0.003   |          |      |
|----------|---------|---------|----------|------|
| females  | 1.7     | 0.002   |          |      |
\-----------------------------------------------/
```

Next, a richer example with grouped rows and columns (adapted from "Developing International Software" by Nadine Kano). First here is what the table looks like on paper:

CODE-PAGE SUPPORT IN MICROSOFT WINDOWS

| Code-Page ID | Name | ACP | OEMCP | Windows NT 3.1 | Windows NT 3.51 | Windows 95 |
|---|---|---|---|---|---|---|
| 1200 | Unicode (BMP of ISO 10646) | | | X | X | * |
| 1250 | Windows 3.1 Eastern European | X | | X | X | X |
| 1251 | Windows 3.1 Cyrillic | X | | X | X | X |
| 1252 | Windows 3.1 US (ANSI) | X | | X | X | X |
| 1253 | Windows 3.1 Greek | X | | X | X | X |
| 1254 | Windows 3.1 Turkish | X | | X | X | X |
| 1255 | Hebrew | X | | | | X |
| 1256 | Arabic | X | | | | X |
| 1257 | Baltic | X | | | | X |
| 1361 | Korean (Johab) | X | | | ** | X |
| 437 | MS-DOS United States | | X | X | X | X |
| 708 | Arabic (ASMO 708) | | X | | | X |
| 709 | Arabic (ASMO 449+, BCON V4) | | X | | | X |
| 710 | Arabic (Transparent Arabic) | | X | | | X |
| 720 | Arabic (Transparent ASMO) | | X | | | X |

The markup for this uses COLGROUP elements to group columns and to set default column alignment. TBODY elements are used to group rows. The FRAME and RULES attributes are used to select which borders to render.

```
<table border=2 frame=hsides rules=groups>
<caption>CODE-PAGE SUPPORT IN MICROSOFT WINDOWS</caption>
<colgroup align=center>
<colgroup align=left>
<colgroup align=center span=2>
<colgroup align=center span=3>
<thead valign=top>
<tr>
<th>Code-Page<br>ID
<th>Name
<th>ACP
<th>OEMCP
<th>Windows<br>NT 3.1
<th>Windows<br>NT 3.51
<th>Windows<br>95
<tbody>
<tr><td>1200<td>Unicode (BMP of ISO 10646)<td><td><td>X<td>X<TD>*
<tr><td>1250<td>Windows 3.1 Eastern European<td>X<td><td>X<td>X<TD>X
<tr><td>1251<td>Windows 3.1 Cyrillic<td>X<td><td>X<td>X<TD>X
<tr><td>1252<td>Windows 3.1 US (ANSI)<td>X<td><td>X<td>X<TD>X
<tr><td>1253<td>Windows 3.1 Greek<td>X<td><td>X<td>X<TD>X
<tr><td>1254<td>Windows 3.1 Turkish<td>X<td><td>X<td>X<TD>X
```

```
<tr><td>1255<td>Hebrew<td>X<td><td><td><td>X
<tr><td>1256<td>Arabic<td>X<td><td><td><td>X
<tr><td>1257<td>Baltic<td>X<td><td><td><td>X
<tr><td>1361<td>Korean (Johab)<td>X<td><td><td>**<td>X
<tbody>
<tr><td>437<td>MS-DOS United States<td><td>X<td>X<td>X<TD>X
<tr><td>708<td>Arabic (ASMO 708)<td><td>X<td><td><td>X
<tr><td>709<td>Arabic (ASMO 449+, BCON V4)<td><td>X<td><td><td>X
<tr><td>710<td>Arabic (Transparent Arabic)<td><td>X<td><td><td>X
<tr><td>720<td>Arabic (Transparent ASMO)<td><td>X<td><td><td>X
</table>
```

---

# Design Rationale

The HTML table model has evolved from studies of existing SGML tables models, the treatment of tables in common word processing packages, and looking at a wide range of tabular layout in magazines, books and other paper-based documents. The model was chosen to allow simple tables to be expressed simply with extra complexity only when needed. This makes it practical to create the markup for HTML tables with everyday text editors and reduces the learning curve for getting started. This feature has been very important to the success of HTML to date.

Increasingly people are using filters from other document formats or direct wysiwyg editors for HTML. It is important that the HTML table model fits well with these routes for authoring HTML. This affects how the representation handles cells which span multiple rows or columns, and how alignment and other presentation properties are associated with groups of cells.

A major consideration for the HTML table model is that the fonts and window sizes etc. in use with browsers are not under the author's control. This makes it risky to rely on column widths specified in terms of absolute units such as picas or pixels. Instead, tables can be dynamically sized to match the current window size and fonts. Authors can provide guidance as to the relative widths of columns, but user agents should to ensure that columns are wide enough to render the width of the largest single element of the cell's content. If the author's specification must be overridden, it is preferred that the relative widths of individual columns are not changed drastically.

For large tables or slow network connections, it is desirable to be able to start displaying the table before all of the data has been received. The default window width for most user agents shows about 80 characters, and the graphics for many HTML pages are designed with these defaults in mind. Authors can provide a hint to user agents to activate incremental display of table contents. This feature requires the author to specify the number of columns, and includes provision for control of table width and the widths of different columns in relative or absolute terms.

For incremental display, the browser needs the number of columns and their widths. The default width of the table is the current window size (`width="100%"`). This can be altered by including a `WIDTH` attribute in the `TABLE` start tag. By default all columns have the same width, but you can specify column widths with one or more `COL` elements before the table data starts.

The remaining issue is the number of columns. Some people have suggested waiting until the first row of the table has been received, but this could take a long time if the cells have a lot of content. On the whole it makes more sense, when incremental display is desired, to get authors to explicitly specify the number of columns in the `TABLE` start tag.

Authors still need a way of informing the browser whether to use incremental display or to automatically size the table to match the cell contents. For the two pass auto sizing mode, the number

of columns is determined by the first pass, while for the incremental mode, the number of columns needs to be stated up front. So it seems to that COLS=*nn* would be better for this purpose than a LAYOUT attribute such as LAYOUT=FIXED or LAYOUT=AUTO.

It is generally held useful to consider documents from two perspectives: Structural idioms such as headers, paragraphs, lists, tables, and figures; and rendering idioms such as margins, leading, font names and sizes. The wisdom of past experience encourages us to separate the structural information in documents from rendering information. Mixing them together ends up causing increased cost of ownership for maintaining documents, and reduced portability between applications and media.

For tables, the alignment of text within table cells, and the borders between cells are, from the purist's point of view, rendering information. In practice, though, it is useful to group these with the structural information, as these features are highly portable from one application to the next. The HTML table model leaves most rendering information to associated style sheets. The model is designed to take advantage of such style sheets but not to require them.

This specification provides a superset of the simpler model presented in earlier work on HTML+. Tables are considered as being formed from an optional caption together with a sequence of rows, which in turn consist of a sequence of table cells. The model further differentiates header and data cells, and allows cells to span multiple rows and columns.

Following the CALS table model, this specification allows table rows to be grouped into head and body and foot sections. This simplifies the representation of rendering information and can be used to repeat table head and foot rows when breaking tables across page boundaries, or to provide fixed headers above a scrollable body panel. In the markup, the foot section is placed before the body sections. This is an optimization shared with CALS for dealing with very long tables. It allows the foot to be rendered without having to wait for the entire table to be processed.

For the visually impaired, HTML offers the hope of setting to rights the damage caused by the adoption of windows based graphical user interfaces. The HTML table model includes attributes for labeling each cell, to support high quality text to speech conversion. The same attributes can also be used to support automated import and export of table data to databases or spreadsheets.

Current desktop publishing packages provide very rich control over the rendering of tables, and it would be impractical to reproduce this in HTML, without making HTML into a bulky rich text format like RTF or MIF. This specification does, however, offer authors the ability to choose from a set of commonly used classes of border styles. The FRAME attribute controls the appearence of the border frame around the table while the RULES attribute determines the choice of rulings within the table. A finer level of control will be supported via rendering annotations. The STYLE attribute can be used for including rendering information with individual elements. Further rendering information can be given with the STYLE element in the document head or via linked style sheets.

During the development of this specification, a number of avenues were investigated for specifying the ruling patterns for tables. One issue concerns the kinds of statements that can be made. Including support for edge subtraction as well as edge addition leads to relatively complex algorithms. For instance work on allowing the full set of table elements to include the FRAME and RULES attributes led to an algorithm involving some 24 steps to determine whether a particular edge of a cell should be ruled or not. Even this additional complexity doesn't provide enough rendering control to meet the full range of needs for tables. The current specification deliberately sticks to a simple intuitive model, sufficient for most purposes. Further experimental work is needed before a more complex approach is standardized.

---

# A walk through the table DTD

The table document type definition provides the formal definition of the allowed syntax for html tables. The following is an annotated listing of the DTD. The complete listing appears at the end of this document.

Note that the TABLE element is a block-like element rather a character-level element. As such it is a peer of other HTML block-like elements such as paragraphs, lists and headers.

# Common Attributes

The following attributes occur in several of the elements and are defined here for brevity. In general, all attribute names and values in this specification are case insensitive, except where noted otherwise. The ID, CLASS and STYLE attributes are required for use with style sheets, while LANG and DIR are needed for internationalization.

```
<!ENTITY % attrs
      "id      ID        #IMPLIED  -- element identifier --
       class   NAMES     #IMPLIED  -- for subclassing elements --
       style   CDATA     #IMPLIED  -- rendering annotation --
       lang    NAME      #IMPLIED  -- as per RFC 1766 --
       dir   (ltr|rtl)   #IMPLIED  -- I18N text direction --">
```

**ID**

Used to define a document-wide identifier. This can be used for naming positions within documents as the destination of a hypertext link. It may also be used by style sheets for rendering an element in a unique style. An ID attribute value is an SGML NAME token. NAME tokens are formed by an initial letter followed by letters, digits, "-" and "." characters. The letters are restricted to A-Z and a-z.

**CLASS**

A space separated list of SGML NAME tokens. CLASS names specify that the element belongs to the corresponding named classes. These may be used by style sheets to provide class dependent renderings.

**STYLE**

A text string providing rendering information specific to this element in a notation independent of HTML. The mechanism for associating rendering information is described in a separate specification.

**LANG**

A LANG attribute identifies the natural language used by the content of the associated element. The syntax and registry of language values are defined by RFC 1766. In summary the language is given as a primary tag followed by zero or more subtags, separated by "-". White space is not allowed and all tags are case insensitive. The name space of tags is administered by IANA. The two letter primary tag is an ISO 639 language abbreviation, while the initial subtag is a two letter ISO 3166 country code. Example values for LANG include:

```
en, en-US, en-uk, i-cherokee, x-pig-latin.
```

**DIR**

Human writing systems are grouped into scripts, which determine amongst other things, the direction the characters are written. Elements of the Latin script are nominally left to right, while those of the Arabic script are nominally right to left. These characters have what is called strong directionality. Other characters can be directionally neutral (spaces) or weak (punctuation).

The DIR attribute specifies an encapsulation boundary which governs the interpretation of neutral and weakly directional characters. It does not override the directionality of strongly directional characters. The DIR attribute value is one of LTR for left to right, or RTL for right to left, e.g. `DIR=RTL`.

When applied to TABLE, it indicates the geometric layout of rows (i.e. row 1 is on right if DIR=RTL, but on the left if DIR=LTR) *and* it indicates a default base directionality for any text in the table's content if no other DIR attribute applies to that text.

---

# Horizontal and Vertical Alignment Attributes

The alignment of cell contents can be specified on a cell by cell basis, or inherited from enclosing elements, such as the row, column or the table element itself.

**ALIGN**

This specifies the horizontal alignment of cell contents.

```
<!-- horizontal alignment attributes for cell contents -->
<!ENTITY % cell.halign
        "align   (left|center|right|justify|char) #IMPLIED
         char    CDATA   #IMPLIED -- alignment char, e.g. char=':' --
         charoff CDATA   #IMPLIED -- offset for alignment char --"
        >
```

The attribute value should be one of LEFT, CENTER, RIGHT, JUSTIFY and CHAR. User agents may treat JUSTIFY as left alignment if they lack support for text justification. ALIGN=CHAR is used for aligning cell contents on a particular character.

For cells spanning multiple rows or columns, where the alignment property is inherited from the row or column, the initial row and column for the cell determines the appropriate alignment property to use.

Note that an alignment attribute on elements within the cell, e.g. on a P element, overrides the normal alignment value for the cell.

**CHAR**

This is used to specify an alignment character for use with align=char, e.g. `char=":"`. The default character is the decimal point for the current language, as set by the LANG attribute. The CHAR attribute value is case sensitive.

**CHAROFF**

Specifies the offset to the first occurrence of the alignment character on each line. If a line doesn't include the alignment character, it should be horizontally shifted to end at the alignment position. The resolved direction of the cell, as determined by the inheritance of the DIR attribute, is used to set whether the offset is from the left or right margin of the cell. For Latin scripts, the offset will be from the left margin, while for Arabic scripts, it will be from the right margin. In addition to standard units, the "%" sign may be used to indicate that the value specifies the alignment position as a percentage offset of the current cell, e.g. `CHAROFF="30%"` indicates the alignment character should be positioned 30% through the cell.

When using the two pass layout algorithm, the default alignment position in the absence of an explicit or inherited `CHAROFF` attribute can be determined by choosing the position that would center lines for which the width before and after the alignment character are at the maximum values for any of the lines in the column for which `ALIGN=CHAR`. For incremental table layout

the suggested default is CHAROFF="50%". If several cells in different rows for the same column use character alignment, then by default, all such cells should line up, regardless of which character is used for alignment. Rules for handling objects too large for column apply when the explicit or implied alignment results in a situation where the data exceeds the assigned width of the column.

**VALIGN**

Defines whether the cell contents are aligned with the top, middle or bottom of the cell.

```
<!-- vertical alignment attributes for cell contents -->
<!ENTITY % cell.valign
        "valign  (top|middle|bottom|baseline)  #IMPLIED"
        >
```

If present, the value of the attribute should be one of: TOP, MIDDLE, BOTTOM or BASELINE. All cells in the same row with valign=baseline should be vertically positioned so that the first text line in each such cell occur on a common baseline. This constraint does not apply to subsequent text lines in these cells.

## Inheritance Order

Alignment properties can be included with most of the table elements: COL, THEAD, TBODY, TFOOT, TR, TH and TD. When rendering cells, horizontal alignment is determined by columns in preference to rows, while for vertical alignment, the rows are more important than the columns. The following table gives the detailed precedence order for each attribute, where X > Y denotes that X takes precedence over Y:

ALIGN, CHAR and CHAROFF:

```
cells > columns > column groups > rows > row groups > default
```

VALIGN, LANG, DIR and STYLE:

```
cells > rows > row groups > columns > column groups > table > default
```

Where cells are defined by TH and TD elements; rows by TR elements; row groups by THEAD, TBODY and TFOOT elements, columns by COL elements; and column groups by COLGROUP and COL elements. Note that there is no inheritance mechanism for the CLASS attribute.

Properties defined on cells take precedence over inherited properties, but are in turn over-ridden by alignment properties on elements within cells. In the absence of an ALIGN attribute along the inheritance path, the recommended default alignment for table cell contents is ALIGN=LEFT for table data and ALIGN=CENTER for table headers. The recommended default for vertical alignment is VALIGN=MIDDLE. These defaults are chosen to match the behaviour of the widely deployed Netscape implementation.

---

# Standard Units for Widths

Several attributes specify widths as a number followed by an optional suffix. The units for widths are specified by the suffix: **pt** denotes points, **pi** denotes picas, **in** denotes inches, **cm** denotes centimeters, **mm** denotes millimeters, **em** denotes em units (equal to the height of the default font), and **px** denotes screen pixels. The default units are screen pixels (chosen for backwards compatibility). The number is an integer value or a real valued number such as "2.5". Exponents, as in "1.2e2", are not allowed. White space is not allowed between the number and the suffix.

The above set of suffices is augmented for certain elements: "%" is used for the WIDTH attribute for the TABLE element. It indicates that the attribute specifies the percentage width of the space between the current left and right margins, e.g. width="50%". For the COL element, "*" is used with the WIDTH attribute to specify relative column widths, e.g. width="3*", using the same representation as the CALS table model.

---

# The TABLE element

```
<!ENTITY % Where "(left|center|right)">

<!ELEMENT table - - (caption?, (col*|colgroup*), thead?, tfoot?, tbody+)>

<!ATTLIST table                     -- table element --
        %attrs;                     -- id, lang, style, dir and class --
        align    %Where;   #IMPLIED -- table position relative to window --
        width    CDATA     #IMPLIED -- table width relative to window --
        cols     NUMBER    #IMPLIED -- used for immediate display mode --
        border   CDATA     #IMPLIED -- controls frame width around table --
        frame    %Frame;   #IMPLIED -- which parts of table frame to include --
        rules    %Rules;   #IMPLIED -- controls rules between cells --
        cellspacing CDATA  #IMPLIED -- spacing between cells --
        cellpadding CDATA  #IMPLIED -- spacing within cells --
        >
```

The TABLE element requires both start and end tags. Table elements start with an optional CAPTION element, optionally followed by either one or more COL elements, or one or more COLGROUP elements, then an optional THEAD, an optional TFOOT, and finally one or more TBODY elements.

---

**ID, CLASS, STYLE, LANG** and **DIR**

See earlier description of common attributes.

**ALIGN**

Defines the horizontal position of the table relative to the current left and right margins. ALIGN=CENTER centers the table midway between the left and right margins. To allow text to flow around the table, use ALIGN=LEFT to position the table at the left margin, with text flowing around its right handside, or use ALIGN=RIGHT to position the table at the right margin, with text flowing around its left handside.

**Note** you can use <BR CLEAR=LEFT> after the table element if you want to avoid text flowing along side the table when you have specified ALIGN=LEFT, or <BR CLEAR=RIGHT> for a right aligned table. Greater control over textflow is possible using style sheets.

**WIDTH**

Specifies the desired width of the table. In addition to the standard units, the "%" sign may used to indicate that the width specifies the percentage width of the space between the current left and right margins, e.g. width="50%". In the absence of this attribute, the table width can be determined by the layout algorithm given later on.

It is recommended that the table width be increased beyond the value indicated by the WIDTH attribute as needed to avoid any overflow of cell contents. Such increases should try to avoid drastic changes to relative column widths specified by the author. To avoid the need for excessive horizontal scrolling, or when such scrolling is impractical or undesired, it may be appropriate to split words across lines.

**COLS**

Specifies the number of columns for the table. If present the user agent may render the table dynamically as data is received from the network without waiting for the complete table to be received. If the WIDTH attribute is missing, a default of "100%" may be assumed for this purpose. If the COLS attribute is absent, a prepass through the table's contents is needed to determine the number of columns together with suitable values for the widths of each column.

**BORDER**

Specifies the width of the border framing the table, see standard units.

**FRAME**

Specifies which sides of the frame to render.

```
<!ENTITY % Frame
   "(void|above|below|hsides|lhs|rhs|vsides|box|border)">
```

VOID
: Don't render any sides of the frame.

ABOVE
: The top side of the frame

BELOW
: The bottom side of the frame

HSIDES
: The top and bottom sides of the frame

LHS
: The left hand side of the frame

RHS
: The right hand side of the frame

VSIDES
: The left and right sides of the frame

BOX
: All four sides of the frame

BORDER
: All four sides of the frame

The value "Border" is included for backwards compatibility with deployed browsers. If a document includes <TABLE BORDER> the user agent will see `FRAME=BORDER` and `BORDER=implied`. If the document includes <TABLE BORDER=*n*> then the user agent should treat this as `FRAME=BORDER` except if *n=0* for which `FRAME=VOID` is appropriate.

*Note: it would have been preferable to choose values for FRAME consistent with the RULES attribute and the values used for alignment. For instance:* `none, top, bottom, topbot, left, right, leftright, all`. *Unfortunately, SGML requires enumerated attribute values to be unique for each element, independent of the attribute name. This causes immediate problems for "none", "left", "right" and "all". The values for FRAME have been chosen to avoid clashes with the RULES, ALIGN and VALIGN attributes. This provides a measure of future proofing, as it is anticipated that that the FRAME and RULES attributes will be added to other table elements in future revisions to this specification. An alternative would be to make FRAME a CDATA attribute. The consensus of the HTML-WG was that the benefits of being able to use SGML validation tools to check attributes based on enumerated values outweighs the need for consistent names.*

**RULES**

Specifies where to draw rules within the table interior.

```
<!ENTITY % Rules "(none | groups | rows | cols | all)">
```

NONE
: Suppresses internal rulings.

GROUPS
: The THEAD, TFOOT and TBODY elements divide the table into groups of rows, while COLGROUP elements divide the table into groups of columns. This choice places a horizontal rule between each row group and a vertical rule between each column group. Note that every table has at least one row and one column group.

ROWS
: As `RULES=GROUPS` plus horizontal rules between all rows. User agents may choose to use a heavier rule between groups of rows and columns for emphasis.

COLS
: As `RULES=GROUPS` plus vertical rules between all columns. User agents may choose to use a heavier rule between groups of rows and columns for emphasis.

ALL
: Place rules between all rows and all columns. User agents may choose to use a heavier rule between groups of rows and columns for emphasis.

If a document includes <TABLE BORDER> or <TABLE BORDER=*n*> then the default for the table element is `RULES=ALL`, except if *n=0* for which `RULES=NONE` is appropriate.

**CELLSPACING**
: This attribute is intended for backwards compatibility with deployed user agents. It specifies the space between the table frame and the first or last cell border for each row or column, and between other cells in the table. See standard units. Greater control will be possible using style sheet languages.

**CELLPADDING**
: This attribute is intended for backwards compatibility with deployed user agents. It specifies the amount of space between the border of the cell and its contents both above/below, and left//right. See standard units. Greater control will be possible using style sheet languages.

If a fixed width is set for the table or column, the CELLSPACING and CELLPADDING may demand more space than assigned. Current practice is for the latter to take precedence over WIDTH attributes when a conflict occurs, although this isn't required by this specification.

---

# Table Captions

```
<!ELEMENT caption - - (%text;)+>

<!ENTITY % Caption "(top|bottom|left|right)">

<!ATTLIST caption                    -- table caption --
        %attrs;                      -- id, lang, style, dir and class --
        align   %Caption; #IMPLIED -- relative to table --
        >
```

The optional CAPTION element is used to provide a caption for the table. Both start and end tags are required.

**ID, CLASS, STYLE, LANG** and **DIR**
: See earlier description of common attributes.

**ALIGN**
: This may be used to control the placement of captions relative to the table. When present, the ALIGN attribute should have one of the values: TOP, BOTTOM, LEFT and RIGHT. It is recommended that the caption is made to fit within the width or height of the table as

appropriate. The default position of the caption is deliberately unspecified.

*Note the ALIGN attribute is overused in HTML, but is retained here for compatibility with currently deployed browsers.*

---

# The COLGROUP Element

```
<!ELEMENT colgroup - O (col*)>

<!ATTLIST colgroup
        %attrs;                     -- id, lang, style, dir and class --
        span    NUMBER    1         -- default number of columns in group --
        width   CDATA     #IMPLIED  -- default width for enclosed COLs --
        %cell.halign;               -- horizontal alignment in cells --
        %cell.valign;               -- vertical alignment in cells --
        >
```

The COLGROUP element acts as a container for a group of columns, and allows you to set default properties for these columns. In the absence of a COLGROUP element, all columns in the table are assumed to belong to a single column group. Each COLGROUP element can contain zero or more COL elements. COLGROUP requires a start tag, but the end tag may be omitted. This is useful when defining a sequence of COLGROUP elements, e.g.

```
<TABLE FRAME=BOX RULES=COLS>
  <COLGROUP>
    <COL WIDTH="1*">
    <COL WIDTH="2*">
  <COLGROUP>
    <COL WIDTH="1*">
    <COL WIDTH="3*">
  <THEAD>
    <TR> ...
</TABLE>
```

COLGROUP elements can be used with the following attributes:

**ID, CLASS, STYLE, LANG** and **DIR**
See earlier description of common attributes.

**SPAN**
A positive integer value that specifies a default for how many columns are in this group. This attribute should be ignored if the COLGROUP element contains one or more COL elements. It provides a convenient way of grouping columns without the need to supply COL elements.

**WIDTH**
Specifies a default width for each of the grouped columns, see standard units. In addition, the "*" suffix denotes relative widths, e.g.

```
width=64        width in screen pixels
width=0.5*      a relative width of 0.5
```

Relative widths act as constraints on the relative widths of different columns. If a COLGROUP element specifies a relative width of zero, all of the columns in the group should be set to their minimum widths, unless they are associated with a COL element with an overriding WIDTH attribute. When widths are given in absolute units, the user agent can use these to constrain the width of the table. The "*" suffix is used to simplify importing tables from the CALS representation.

**ALIGN, CHAR, CHAROFF** and **VALIGN**

Specify values for horizontal and vertical alignment within table cells. See inheritance order of alignment properties.

---

# The COL Element

```
<!ELEMENT col - O EMPTY>

<!ATTLIST col                       -- column groups and properties --
        %attrs;                     -- id, lang, style, dir and class --
        span    NUMBER    1         -- number of columns spanned by group --
        width   CDATA     #IMPLIED  -- column width specification --
        %cell.halign;               -- horizontal alignment in cells --
        %cell.valign;               -- vertical alignment in cells --
        >
```

This optional element is used to specify column based defaults for table properties. It is an empty element, and as such has no content, and shouldn't be given an end tag. Several COL elements may be given in succession. COL attributes override those of the parent COLGROUP element.

**ID, CLASS, STYLE, LANG** and **DIR**

See earlier description of common attributes.

**SPAN**

A positive integer value that specifies how many columns this element applies to, defaulting to one. In the absence of SPAN attributes the first COL element applies to the first column, the second COL element to the second column and so on. If the second COL element had SPAN=2, it would apply to the second and third column. The next COL element would then apply to the fourth column and so on. SPAN=0 has a special significance and implies that the COL element spans all columns from the current column up to and including the last column. Note that a COL SPAN does not define a group. It is merely a way to share attribute definitions.

**WIDTH**

Specifies the width of the columns, see standard units. If the element spans several columns then the `WIDTH` attribute specifies the width for each of the individual columns - not the width of the span. In addition, the "*" suffix denotes relative widths, e.g.

```
width=64        width in screen pixels
width=0.5*      a relative width of 0.5
```

Relative widths act as constraints on the relative widths of different columns. If a COL element specifies a relative width of zero, the column should always be set to its minimum width. When widths are given in absolute units, the user agent can use these to constrain the width of the table. The "*" suffix is used to simplify importing tables from the CALS representation.

**ALIGN, CHAR, CHAROFF** and **VALIGN**

Specify values for horizontal and vertical alignment within table cells. See inheritance order of alignment properties.

---

# Table Head, Foot and Body Elements

```
<!ELEMENT thead - O tr+>
<!ELEMENT tfoot - O tr+>
<!ELEMENT tbody O O tr+>
```

```
<!ATTLIST (thead|tbody|tfoot)      -- table section --
        %attrs;                    -- id, lang, style, dir and class --
        %cell.halign;              -- horizontal alignment in cells --
        %cell.valign;              -- vertical alignment in cells --
        >
```

Tables may be divided up into head and body sections. The THEAD and TFOOT elements are optional, but one or more TBODY elements are always required. If the table only consists of a TBODY section, the TBODY start and end tags may be omitted, as the parser can infer them. If a THEAD element is present, the THEAD start tag is required, but the end tag can be omitted, provided a TFOOT or TBODY start tag follows. The same applies to TFOOT.

*Note: This definition provides compatibility with tables created for the older model, as well as allowing the end tags for THEAD, TFOOT and TBODY to be omitted.*

The THEAD, TFOOT and TBODY elements provide a convenient means for controlling rendering. If the table has a large number of rows in the body, user agents may choose to use a scrolling region for the table body sections. When rendering to a paged device, tables will often have to be broken across page boundaries. The THEAD, TFOOT and TBODY elements allow the user agent to repeat the table foot at the bottom of the current page, and then the table head at the top of the new page before continuing on with the table body.

TFOOT is placed before the TBODY in the markup sequence, so that browsers can render the foot before receiving all of the table data. This is useful when very long tables are rendered with scrolling body sections, or for paged output, involving breaking the table over many pages.

Each THEAD, TFOOT and TBODY element must contain one or more TR elements.

**ID, CLASS, STYLE, LANG** and **DIR**
See earlier description of common attributes.

**ALIGN, CHAR, CHAROFF** and **VALIGN**
Specify values for horizontal and vertical alignment within table cells. See inheritance order of alignment properties.

---

# Table Row (TR) elements

```
<!ELEMENT tr - O (th|td)+>

<!ATTLIST tr                       -- table row --
        %attrs;                    -- id, lang, style, dir and class --
        %cell.halign;              -- horizontal alignment in cells --
        %cell.valign;              -- vertical alignment in cells --
        >
```

The TR or table row element acts as a container for a row of table cells. The end tag may be omitted.

**ID, CLASS, STYLE, LANG** and **DIR**
See earlier description of common attributes.

**ALIGN, CHAR, CHAROFF** and **VALIGN**
Specify values for horizontal and vertical alignment within table cells. See inheritance order of alignment properties.

---

# Table Cells: TH and TD

```
<!ELEMENT (th|td) - O %body.content>

<!ATTLIST (th|td)                    -- header or data cell --
        %attrs;                      -- id, lang, style, dir and class --
        axis    CDATA    #IMPLIED    -- defaults to cell content --
        axes    CDATA    #IMPLIED    -- list of axis names --
        nowrap (nowrap)  #IMPLIED    -- suppress word wrap --
        rowspan NUMBER   1           -- number of rows spanned by cell --
        colspan NUMBER   1           -- number of cols spanned by cell --
        %cell.halign;                -- horizontal alignment in cells --
        %cell.valign;                -- vertical alignment in cells --
        >
```

TH elements are used to represent header cells, while TD elements are used to represent data cells. This allows user agents to render header and data cells distinctly, even in the absence of style sheets.

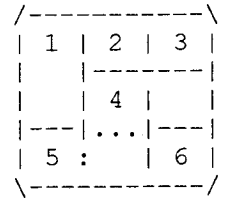
Cells can span multiple rows and columns, and may be empty. Cells spanning rows contribute to the column count on each of the spanned rows, but only appear in the markup once (in the first row spanned). The row count is determined by the number of TR elements. Any rows implied by cells spanning rows beyond this should be ignored.

If the column count for the table is greater than the number of cells for a given row (after including cells for spanned rows), the missing cells are treated as occurring on the right hand side of the table and rendered as empty cells. If the language context indicates a right to left writing order, then the missing cells should be placed on the left hand side.

It is possible to create tables with overlapping cells, for instance:

```
<table border>
<tr><td rowspan=2>1<td>2<td>3
<tr><td rowspan=2>4
<tr><td colspan=2>5<td>6
</table>
```

which might look something like:




In this example, the cells labelled 4 and 5 overlap. In such cases, the rendering is implementation dependent.

The AXIS and AXES attributes for cells provide a means for defining concise labels for cells. When rendering to speech, these attributes may be used to provide abbreviated names for the headers relevant to each cell. Another application is when you want to be able to later process table contents to enter them into a database. These attributes are then used to give database field names. The table's class attribute should be used to let the software recognize which tables can be treated in this way.

**ID, CLASS, STYLE, LANG** and **DIR**

See earlier description of common attributes.

**AXIS**
This defines an abbreviated name for a header cell, e.g. which can be used when rendering to speech. It defaults to the cell's content.

**AXES**
This is a comma separated list of axis names which together identify the row and column headers that pertain to this cell. It is used for example when rendering to speech to identify the cell's position in the table. If missing the user agent can try to follow up columns and left along rows (right for some languages) to find the corresponding header cells.

**NOWRAP**, e.g. <TD NOWRAP>
The presence of this attribute disables automatic wrapping of text lines for this cell. If used uncautiously, it may result in excessively wide cells. This attribute is defined for backwards compatibility with deployed user agents. Greater control is possible with associated style sheet languages (for example for control over overflow handling).

**ROWSPAN**, e.g. <TD ROWSPAN=2>
A positive integer value that defines how may rows this cell spans. The default ROWSPAN is 1. ROWSPAN=0 has a special significance and implies that the cell spans all rows from the current row up to the last row of the table.

**COLSPAN**, e.g. <TD COLSPAN=2>
A positive integer value that defines how may columns this cell spans. The default COLSPAN is 1. COLSPAN=0 has a special significance and implies that the cell spans all columns from the current column up to the last column of the table.

**ALIGN, CHAR, CHAROFF** and **VALIGN**
Specify values for horizontal and vertical alignment within table cells. See inheritance order of alignment properties.

**Note:** It is recommended that implementors provide support for the Netscape 1.1 `WIDTH` attribute for `TH` and `TD`, although this isn't part of the current specification. Document authors are advised to use the width attribute for the `COL` element instead.

---

# Recommended Layout Algorithms

If the COLS attribute on the TABLE element specifies the number of columns, then the table may be rendered using a fixed layout, otherwise the autolayout algorithm described below should be used.

## Fixed Layout Algorithm

For this algorithm, it is assumed that the number of columns is known. The column widths by default should be set to the same size. Authors may override this by specifying relative or absolute column widths, using the COLGROUP or COL elements. The default table width is the space between the current left and right margins, but may be overridden by the WIDTH attribute on the TABLE element, or determined from absolute column widths. To deal with mixtures of absolute and relative column widths, the first step is to allocate space from the table width to columns with absolute widths. After this, the space remaining is divided up between the columns with relative widths.

The table syntax alone is insufficient to guarantee the consistency of attribute values. For instance, the number of columns specified by the COLS attribute may be inconsistent with the number of columns implied by the COL elements. This in turn, may be inconsistent with the number of columns implied by the table cells. A further problem occurs when the columns are too narrow to avoid overflow of cell contents. The width of the table as specified by the TABLE element or COL elements may result in overflow of cell contents. It is recommended that user agents attempt to recover gracefully from these situations, e.g. by hyphenating words and resorting to splitting words if hyphenation points are

unknown.

In the event that an indivisible element causes cell overflow, the user agent may consider adjusting column widths and re-rendering the table. In the worst case clipping may be considered if column width adjustments and/or scrollable cell content are not feasible. In any case if cell content is split or clipped this should be indicated to the user in an appropriate manner.

## Autolayout Algorithm

If the COLS attribute is missing from the table start tag, then the user agent should use the following autolayout algorithm. It uses two passes through the table data and scales linearly with the size of the table.

In the first pass, line wrapping is disabled, and the user agent keeps track of the minimum and maximum width of each cell. The maximum width is given by the widest line. As line wrap has been disabled, paragraphs are treated as long lines unless broken by <BR> elements. The minimum width is given by the widest word or image etc. taking into account leading indents and list bullets etc. In other words, if you were to format the cell's content in a window of its own, determine the minimum width you could make the window before the cell begins to overflow. Allowing user agents to split words will minimize the need for horizontal scrolling or in the worst case clipping of cell contents.

This process also applies to any nested tables occuring in cell content. The minimum and maximum widths for cells in nested tables are used to determine the minimum and maximum widths for these tables and hence for the parent table cell itself. The algorithm is linear with aggregate cell content, and broadly speaking independent of the depth of nesting.

To cope with character alignment of cell contents, the algorithm keeps three running min/max totals for each column: Left of align char, right of align char and un-aligned. The minimum width for a column is then: `max(min_left + min_right, min_non-aligned)`.

The minimum and maximum cell widths are then used to determine the corresponding minimum and maximum widths for the columns. These in turn, are used to find the minimum and maximum width for the table. Note that cells can contain nested tables, but this doesn't complicate the code significantly. The next step is to assign column widths according to the available space (i.e. the space between the current left and right margins).

For cells which span multiple columns, a simple approach, as used by Arena, is to evenly apportion the min/max widths to each of the constituent columns. A slightly more complex approach is to use the min/max widths of unspanned cells to weight how spanned widths are apportioned. Experimental study suggests a blend of the two approaches will give good results for a wide range of tables.

The table borders and intercell margins need to be included in assigning column widths. There are three cases:

1. **The minimum table width is equal to or wider than the available space.** In this case, assign the minimum widths and allow the user to scroll horizontally. For conversion to braille, it will be necessary to replace the cells by references to notes containing their full content. By convention these appear before the table.
2. **The maximum table width fits within the available space.** In this case, set the columns to their maximum widths.
3. **The maximum width of the table is greater than the available space, but the minimum table width is smaller.** In this case, find the difference between the available space and the minimum table width, lets call it **W**. Lets also call **D** the difference between maximum and

minimum width of the table.

For each column, let **d** be the difference between maximum and minimum width of that column. Now set the column's width to the minimum width plus **d** times **W** over **D**. This makes columns with large differences between minimum and maximum widths wider than columns with smaller differences.

This assignment step is then repeated for nested tables using the minimum and maximum widths derived for all such tables in the first pass. In this case, the width of the parent (i.e. enclosing) table cell plays the role of the current window size in the above description. This process is repeated recursively for all nested tables. The topmost table is then rendered using the assigned widths. Nested tables are subsequently rendered as part of the parent table's cell contents.

If the table width is specified with the WIDTH attribute, the user agent attempts to set column widths to match. The WIDTH attribute is not binding if this results in columns having less than their minimum (i.e. indivisible) widths.

If relative widths are specified with the COL element, the algorithm is modified to increase column widths over the minimum width to meet the relative width constraints. The COL elements should be taken as hints only, so columns shouldn't be set to less than their minimum width. Similarly, columns shouldn't be made so wide that the table stretches well beyond the extent of the window. If a COL element specifies a relative width of zero, the column should always be set to its minimum width.

---

# HTML Table DTD

The DTD or document type definition provides the formal definition of the allowed syntax for HTML tables.

```
<!-- Content model entities imported from parent DTD:

  %body.content; allows table cells to contain headers, paras,
  lists, form elements and even arbitrarily nested tables.

  %text; is text characters, including character entities and
  character emphasis elements, IMG and anchors
-->

<!ENTITY % attrs
      "id      ID        #IMPLIED  -- element identifier --
       class   NAMES     #IMPLIED  -- for subclassing elements --
       style   CDATA     #IMPLIED  -- rendering annotation --
       lang    NAME      #IMPLIED  -- as per RFC 1766 --
       dir   (ltr|rtl)   #IMPLIED  -- I18N text direction --">

<!--
 The BORDER attribute sets the thickness of the frame around the
 table. The default units are screen pixels.

 The FRAME attribute specifies which parts of the frame around
 the table should be rendered. The values are not the same as
 CALS to avoid a name clash with the VALIGN attribute.

 The value "border" is included for backwards compatibility with
 <TABLE BORDER> which yields frame=border and border=implied
 For <TABLE BORDER=1> you get border=1 and frame=implied. In this
 case, its appropriate to treat this as frame=border for backwards
 compatibility with deployed browsers.
-->
```

```
<!ENTITY % Frame "(void|above|below|hsides|lhs|rhs|vsides|box|border)">

<!--
 The RULES attribute defines which rules to draw between cells:

 If RULES is absent then assume:
     "none" if BORDER is absent or BORDER=0 otherwise "all"
-->

<!ENTITY % Rules "(none | groups | rows | cols | all)">

<!-- horizontal placement of table relative to window -->
<!ENTITY % Where "(left|center|right)">
<!-- horizontal alignment attributes for cell contents -->
<!ENTITY % cell.halign
        "align  (left|center|right|justify|char) #IMPLIED
         char    CDATA   #IMPLIED -- alignment char, e.g. char=':' --
         charoff CDATA   #IMPLIED -- offset for alignment char --"
        >

<!-- vertical alignment attributes for cell contents -->
<!ENTITY % cell.valign
        "valign  (top|middle|bottom|baseline)  #IMPLIED"
        >

<!ELEMENT table - - (caption?, (col*|colgroup*), thead?, tfoot?, tbody+)>
<!ELEMENT caption - - (%text;)+>
<!ELEMENT thead - O (tr+)>
<!ELEMENT tfoot - O (tr+)>
<!ELEMENT tbody O O (tr+)>
<!ELEMENT colgroup - O (col*)>
<!ELEMENT col - O EMPTY>
<!ELEMENT tr - O (th|td)+>
<!ELEMENT (th|td) - O %body.content>

<!ATTLIST table                     -- table element --
        %attrs;                     -- id, lang, style, dir and class --
        align   %Where;  #IMPLIED   -- table position relative to window --
        width   CDATA    #IMPLIED   -- table width relative to window --
        cols    NUMBER   #IMPLIED   -- used for immediate display mode --
        border  CDATA    #IMPLIED   -- controls frame width around table --
        frame   %Frame;  #IMPLIED   -- which parts of table frame to include --
        rules   %Rules;  #IMPLIED   -- rulings between rows and cols --
        cellspacing CDATA #IMPLIED -- spacing between cells --
        cellpadding CDATA #IMPLIED -- spacing within cells --
        >

<!-- ALIGN is used here for compatibility with deployed browsers -->
<!ENTITY % Caption "(top|bottom|left|right)">

<!ATTLIST caption                   -- table caption --
        %attrs;                     -- id, lang, style, dir and class --
        align  %Caption; #IMPLIED   -- relative to table --
        >

<!--
COLGROUP groups a set of COL elements. It allows you to group
several columns together.
-->

<!ATTLIST colgroup
        %attrs;                     -- id, lang, style, dir and class --
        span    NUMBER   1          -- default number of columns in group --
        width   CDATA    #IMPLIED   -- default width for enclosed COLs --
        %cell.halign;               -- horizontal alignment in cells --
        %cell.valign;               -- vertical alignment in cells --
```

```
        >

<!--
 COL elements define the alignment properties for cells in a given
 column or spanned columns. The WIDTH attribute specifies the
 width of the columns, e.g.

     width=64        width in screen pixels
     width=0.5*      relative width of 0.5
-->

<!ATTLIST col                       -- column groups and properties --
        %attrs;                     -- id, lang, style, dir and class --
        span    NUMBER   1          -- number of columns spanned by group --
        width   CDATA    #IMPLIED   -- column width specification --
        %cell.halign;               -- horizontal alignment in cells --
        %cell.valign;               -- vertical alignment in cells --
        >

<!--
    Use THEAD to duplicate headers when breaking table
    across page boundaries, or for static headers when
    body sections are rendered in scrolling panel.

    Use TFOOT to duplicate footers when breaking table
    across page boundaries, or for static footers when
    body sections are rendered in scrolling panel.

    Use multiple TBODY sections when rules are needed
    between groups of table rows.
-->
<!ATTLIST (thead|tbody|tfoot)       -- table section --
        %attrs;                     -- id, lang, style, dir and class --
        %cell.halign;               -- horizontal alignment in cells --
        %cell.valign;               -- vertical alignment in cells --
        >

<!ATTLIST tr                        -- table row --
        %attrs;                     -- id, lang, style, dir and class --
        %cell.halign;               -- horizontal alignment in cells --
        %cell.valign;               -- vertical alignment in cells --
        >

<!ATTLIST (th|td)                   -- header or data cell --
        %attrs;                     -- id, lang, style, dir and class --
        axis    CDATA    #IMPLIED   -- defaults to cell content --
        axes    CDATA    #IMPLIED   -- list of axis names --
        nowrap (nowrap)  #IMPLIED   -- suppress word wrap --
        rowspan NUMBER   1          -- number of rows spanned by cell --
        colspan NUMBER   1          -- number of cols spanned by cell --
        %cell.halign;               -- horizontal alignment in cells --
        %cell.valign;               -- vertical alignment in cells --
        >
```

---

# References

**Arena**

W3C's HTML3 browser, see "http://www.w3.org/pub/WWW/Arena/". Arena was originally created as a proof of concept demo for ideas in the HTML+ specification that preceded HTML3. The browser is now being re-implemented to provide a reference implementation of HTML3 along with support for style sheets and client-side scripting.

**CALS**

Continuous Acquisition and Life-Cycle Support (formerly Computer-aided Acquisition and Logistics Support) (CALS) is a Department of Defense (DoD) strategy for achieving effective creation, exchange, and use of digital data for weapon systems and equipment. More information can be found from the US Navy CALS home page at http://navysgml.dt.navy.mil/cals.html

**HTML 3.0**

HyperText Markup Language Specification Version 3.0. This is the initial draft specification as published in March 1995. Work on refining HTML3 is proceeding piecemeal with the new table specification as one of the pieces. For W3C related work on HTML, see "http://www.w3.org/pub/WWW/MarkUp/".

**RFC 1766**

"Tags for the Identification of Languages", by H. Alvestrand, UNINETT, March 1995. This document can be downloaded from "ftp://ds.internic.net/rfc/rfc1766.txt".

**HTML and Style Sheets**

Associating HTML3 documents with Rendering Information, by Bert Bos, Dave Raggett and Håkon Lie. This document can be downloaded from: "http://www.w3.org/pub/WWW/TR/WD-style.html"

---

The World Wide Web Consortium: http://www.w3.org/

# Special Characters

HTML allows for the insertion of any character defined in the ISO 8859-1 character set, into Web pages. These characters may not always be visible to all browsers, but they are defined as available entities. Table A lists all the special characters with their corresponding HTML code. If you want to put a particular character in your Web page, simply type in the entire string under the "HTML Code" heading. Some of the characters have two possible HTML codes, so simply use one of them. Be sure to include the ampersand (&) before the code, and the semicolon (;) after the code.

**Table A.1 HTML Codes for ISO 8859-1 Characters**

| Description of Character | HTML Code | Example of Character |
|---|---|---|
| Quotation mark | &quot; | " |
| Ampersand | &amp; | & |
| Less-than sign | &lt; | < |
| Greater-than sign | &gt; | > |
| Non-breaking space |   | |
| Inverted exclamation mark | &iexcl; | ¡ |
| Cent sign | &cent; | ¢ |

(continues)

Special characters &#nnn;

nnn (decimal) code eg 169 = copyright sign ©

**Table A.1 Continued**

| Description of Character | HTML Code | Example of Character |
|---|---|---|
| Pound sterling | £ | £ |
| General currency sign | ¤ | ¤ |
| Yen sign | ¥ | ¥ |
| Broken vertical bar | ¦ or &brkbar; | ¦ |
| Section sign | § | § |
| Umlaut (dieresis) | ¨ or ¨ | ¨ |
| Copyright | © | © |
| Feminine ordinal | ª | ª |
| Left angle quote (guillemotleft) | « | « |
| Not sign | ¬ | ¬ |
| Soft hyphen |  | - |
| Registered trademark | ® | ® |
| Macron accent | ¯ or &hibar; | ¯ |
| Degree sign | ° | ° |
| Plus or minus | ± | ± |
| Superscript two | ² | ² |
| superscript three | ³ | ³ |
| Acute accent | ´ | ´ |
| Micro sign | µ | µ |
| Paragraph sign | ¶ | ¶ |
| Middle dot | · | · |
| Cedilla | ¸ | ¸ |
| Superscript one | ¹ | ¹ |
| Masculine ordinal | º | º |
| Right angle quote (guillemotright) | » | » |
| Fraction one-fourth | ¼ | ¼ |
| Fraction one-half | ½ | ½ |
| Fraction three-fourths | ¾ | ¾ |
| Inverted question mark | ¿ | ¿ |
| capital A, grave accent | À | À |
| Capital A, acute accent | Á | Á |
| Capital A, circumflex accent | Â | Â |

| Description of Character | HTML Code | Example of Character |
|---|---|---|
| Capital A, tilde | Ã | Ã |
| Capital A, umlaut (dieresis) | Ä | Ä |
| Capital A, ring | Å | Å |
| Capital AE, diphthong (ligature) | &Aelig; | Æ |
| Capital C, cedilla | Ç | Ç |
| Capital E, grave accent | È | È |
| Capital E, acute accent | É | É |
| Capital E, circumflex accent | Ê | Ê |
| Capital E, umlaut (dieresis) | Ë | Ë |
| Capital I, grave accent | Ì | Ì |
| Capital I, acute accent | Í | Í |
| Capital I, circumflex accent | Î | Î |
| Capital I, umlaut (dieresis) | Ï | Ï |
| Capital Eth, Icelandic | Ð or Đ | Ð |
| Capital N, tilde | Ñ | Ñ |
| Capital O, grave accent | Ò | Ò |
| Capital O, acute accent | Ó | Ó |
| Capital O, circumflex accent | Ô | Ô |
| Capital O, tilde | Õ | Ö |
| Capital O, umlaut (dieresis) | Ö | Ö |
| Multiply sign | × | x |
| Capital O, slash | Ø | Ø |
| Capital U, grave accent | Ù | Ù |
| Capital U, acute accent | Ú | Ú |
| Capital U, circumflex accent | Û | Û |
| Capital U, umlaut (dieresis) | Ü | Ü |
| Capital Y, acute accent | Ý | Y |
| Capital THORN, Icelandic | Þ | Þ |
| Small sharp s, German (sz ligature) | ß | ß |
| Small a, grave accent | à | à |
| Small a, acute accent | á | á |
| Small a, circumflex accent | â | â |
| Small a, tilde | ã | ã |
| Small a, umlaut (dieresis) | ä | ä |

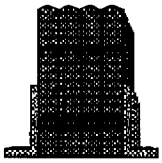
(continues)

**Table A.1 Continued**

| Description of Character | HTML Code | Example of Char |
|---|---|---|
| Small a, ring | å | å |
| Small ae diphthong (ligature) | æ | æ |
| Small c, cedilla | ç | ç |
| Small e, grave accent | è | è |
| Small e, acute accent | é | é |
| Small e, circumflex accent | ê | ê |
| Small e, umlaut (dieresis) | ë | ë |
| Small i, grave accent | ì | ì |
| Small i, acute accent | í | í |
| Small i, circumflex accent | î | î |
| Small i, umlaut (dieresis) | ï | ï |
| Small eth, Icelandic | ð | |
| Small n, tilde | ñ | ñ |
| Small o, grave accent | ò | ò |
| Small o, acute accent | ó | ó |
| Small o, circumflex accent | ô | ô |
| Small o, tilde | õ | õ |
| Small o, umlaut (dieresis) | ö | ö |
| Division sign | ÷ | ÷ |
| Small o, slash | ø | ø |
| Small u, grave accent | ù | ù |
| Small u, acute accent | ú | ú |
| Small u, circumflex accent | û | û |
| Small u, umlaut (dieresis) | ü | ü |
| Small y, acute accent | ý | ý |
| Small thorn, Icelandic | þ | þ |
| Small y, umlaut (dieresis) | ÿ | ÿ |

# Order Entry Form

This is sample order entry form with default values for some text entry fields.



## CUSTOMER

Last Name:
First Name:
Middle Initial:
Phone Number:
Street Address:
City:
State:
Zip:

## ORDER

Quantity: CS6400-4 CS6400 SuperServer (4 Processors)
Quantity: CS6400-4 CS6400 SuperServer (4 Processors)
Quantity: CS6400-4 CS6400 SuperServer (4 Processors)

## PAYMENT


Method of Payment: American Express
Card Number:
Expiration Date:

Place Order Reset to Default

```
        <OPTION> TS6400-8 CS6400 SuperServer (8 Processors)
        </SELECT></TD></TR>
<TR><TD>Quantity:</TD>
    <TD><INPUT TYPE=TEXT NAME="QTY3" SIZE=5></TD>
    <TD><SELECT NAME="ORDERITEM3">
        <OPTION> CS6400-4 CS6400 SuperServer (4 Processors)
        <OPTION> CS6400-8 CS6400 SuperServer (8 Processors)
        <OPTION> CS6400-64 CS6400 SuperServer (64 Processors)
        </SELECT></TD></TR>
</TABLE>
<HR>

<H2><CENTER> PAYMENT </CENTER></H2>

<CENTER>
<img src=/mapper-image/VISA.GIF>
<img src=/mapper-image/MCARD.GIF>
<img src=/mapper-image/AMEX.GIF>
</CENTER><P>

<TABLE ALIGN=CENTER BORDER=0 BGCOLOR=#008080>
<TR><TD>Method of Payment:</TD>
    <TD><SELECT NAME="PAYMENTTYPE">
        <OPTION SELECTED> American Express
        <OPTION> MasterCard
        <OPTION> Visa
        </SELECT></TD></TR>

<TR><TD>Card Number:</TD>
    <TD><INPUT NAME="CARDNUMBER" size=20 maxlength=20></TD></TR
>
<TR><TD>Expiration Date:</TD>
    <TD><INPUT NAME="EXPDATE" size=5 maxlength=5></TD></TR>
</TABLE>

<P>
<CENTER>
<INPUT TYPE="submit" VALUE="Place Order">
<INPUT TYPE="reset" VALUE="Reset to Default"> <BR>
</CENTER>

</FORM>
 <HR><FONT FACE="ARIAL" SIZE="1" COLOR=#0000FF>
 &#169; 1996 The Internet Commerce Enabler - A Solution from UN
ISYS</FONT>
</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE> Order Entry Form </TITLE>
</HEAD>
<BODY TEXT=#000000 BGCOLOR=#C0C0C0>

<H1><CENTER> Order Entry Form </CENTER></H1>

This is sample order entry form with default values for some
text entry fields. <P>
<IMG SRC="/mapper-image/BOOKS.GIF" WIDTH=86 HEIGHT=86 BORDER=0>

<HR>

<FORM METHOD="POST" ACTION="/cgi/ice.cgi/UserId=GUEST/Dept=7/Pa
ssWd=/Category=Forms/Service=CONFIRM1">

<H2><CENTER> CUSTOMER </CENTER></H2>

<TABLE ALIGN=CENTER BORDER=0 BGCOLOR=#008080>
<TR><TD>Last Name:</TD>
    <TD><INPUT NAME="NAMELAST" size=40 maxlength=40></TD></TR>
<TR><TD>First Name:</TD>
    <TD><INPUT NAME="NAMEFIRST" size=40 maxlength=40></TD></TR>
<TR><TD>Middle Initial:</TD>
    <TD><INPUT NAME="NAMEMI" size=1 maxlength=1></TD></TR>
<TR><TD>Phone Number:</TD>
    <TD><INPUT NAME="NAMEPHONE" size=15 maxlength=15></TD></TR>
<TR><TD>Street Address:</TD>
    <TD><INPUT NAME="NAMESTREET" size=40 maxlength=40></TD></TR
>
<TR><TD>City:</TD>
    <TD><INPUT NAME="NAMECITY" size=20 maxlength=20></TD></TR>
<TR><TD>State:</TD>
    <TD><INPUT NAME="NAMESTATE" size=3 maxlength=3></TD></TR>
<TR><TD>Zip:</TD>
    <TD><INPUT NAME="NAMEZIP" size=9 maxlength=9></TD></TR>
</TABLE>
<HR>

<H2><CENTER> ORDER </CENTER></H2>

<TABLE ALIGN=CENTER BORDER=0 BGCOLOR=#008080>
<TR><TD>Quantity:</TD>
    <TD><INPUT TYPE=TEXT NAME="QTY1" SIZE=5></TD>
    <TD><SELECT NAME="ORDERITEM1">
        <OPTION> CS6400-4 CS6400 SuperServer (4 Processors)
        <OPTION> CS6400-8 CS6400 SuperServer (8 Processors)
        <OPTION> TS6400-8 CS6400 SuperServer (8 Processors)
        </SELECT></TD></TR>
<TR><TD>Quantity:</TD>
    <TD><INPUT TYPE=TEXT NAME="QTY2" SIZE=5></TD>
    <TD><SELECT NAME="ORDERITEM2">
        <OPTION> CS6400-4 CS6400 SuperServer (4 Processors)
        <OPTION> CS6400-8 CS6400 SuperServer (8 Processors)
```

# Fill-Out Form Example #7

This is another fill-out form example, with toggle buttons.

---

## Godzilla's Pizza -- Internet Delivery Service, Part II

Type in your street address: [ ]

Type in your phone number: [ ]

Which toppings would you like?

1. [x] Pepperoni.
2. [ ] Sausage.
3. [ ] Anchovies.

How would you like to pay? Choose any one of the following:

1. ( ) Cash.
2. (•) Check.
3. *Credit card:*
    - ( ) Mastercard.
    - ( ) Visa.
    - ( ) American Express.

Would you like the driver to call before leaving the store?

- (•) *Yes.*
- ( ) *No.*

To order your pizza, press this button: [ Order Pizza ]

---

Things you may want to note:

- Radio buttons (type `RADIO`) are one-of-many input elements. The "many" is all radio buttons in the form with the same `NAME`.
- In other ways, `RADIO` is the same as `CHECKBOX`.
- Multiple sets of radio buttons (radio buttons with the same `NAME`) can be in a single form. This may require careful document/interface design to keep things obvious -- don't intermix radio buttons of different `NAME` values, or the user won't understand how the interface works.

Back to example 6 or forward to example 8.

---

```
<HTML>
<HEAD>
<TITLE>Fill-Out Form Example #7</TITLE>
</HEAD>
<BODY TEXT=#000000 BGCOLOR=#C0C0C0>
<H1>Fill-Out Form Example #7</H1>

This is another fill-out form example, with toggle buttons. <P>

<HR>

<FORM METHOD="POST" ACTION="/cgi/ice.cgi/UserId=GUEST/Dept=7/Pa
ssWd=/Category=Examples/Service=POST-QUERY">

<H2>Godzilla&#39;s Pizza -- Internet Delivery Service, Part II<
/H2>

Type in your street address: <INPUT NAME="address"> <P>

Type in your phone number: <INPUT NAME="phone"> <P>

Which toppings would you like? <P>

<OL>
<LI> <INPUT TYPE="checkbox" NAME="topping1" VALUE="pepperoni">
     Pepperoni.
<LI> <INPUT TYPE="checkbox" NAME="topping2" VALUE="sausage"> Sa
usage.
<LI> <INPUT TYPE="checkbox" NAME="topping3" VALUE="anchovies">
     Anchovies.
</OL>

How would you like to pay?  Choose any one of the following: <P
>

<OL>
<LI> <INPUT TYPE="radio" NAME="paymethod" VALUE="cash" CHECKED>
 Cash.
<LI> <INPUT TYPE="radio" NAME="paymethod" VALUE="check"> Check.
<LI> <I>Credit card:</I>
<UL>
<LI> <INPUT TYPE="radio" NAME="paymethod" VALUE="mastercard"> M
astercard.
<LI> <INPUT TYPE="radio" NAME="paymethod" VALUE="visa"> Visa.
<LI> <INPUT TYPE="radio" NAME="paymethod" VALUE="americanexpres
s">
     American Express.
</UL>
</OL>

Would you like the driver to call before leaving the store? <P>

<DL>
<DD> <INPUT TYPE="radio" NAME="callfirst" VALUE="yes" CHECKED>
<I>Yes.</I>
```

```
<DD> <INPUT TYPE="radio" NAME="callfirst" VALUE="no"> <I>No.</I
>
</DL>


To order your pizza, press this button: <INPUT TYPE="submit"
VALUE="Order Pizza">. <P>

</FORM>

<HR> <P>

Things you may want to note: <P>

<UL>
<LI> Radio buttons (type <CODE>RADIO</CODE>) are one-of-many in
put
     elements.  The "many" is all radio buttons in the form wit
h the
     same <CODE>NAME</CODE>.
<LI> In other ways, <CODE>RADIO</CODE> is the same as
     <CODE>CHECKBOX</CODE>.
<LI> Multiple sets of radio buttons (radio buttons with the sam
e
     <CODE>NAME</CODE>) can be in a single form.  This may requ
ire
     careful document/interface design to keep things obvious -
- don&#39;t
     intermix radio buttons of different <CODE>NAME</CODE> valu
es, or
     the user won&#39;t understand how the interface works.
</UL>

<A HREF="/cgi/ice.cgi/UserId=GUEST/Dept=7/PassWd=/Category=Exam
ples/Service=FRMEX06">Back to example 6</A> or
<A HREF="/cgi/ice.cgi/UserId=GUEST/Dept=7/PassWd=/Category=Exam
ples/Service=FRMEX08">forward to example 8</A>. <P>
 <HR><FONT FACE="ARIAL" SIZE="1" COLOR=#0000FF>
 &#169; 1996 The Internet Commerce Enabler - A Solution from UN
ISYS</FONT>
</BODY>
</HTML>
```

CHAPTER 8

# Publishing Information and Applications



Peer Web Services for Windows NT can publish both information and applications. This means that your Web site can contain anything from static pages of information to interactive applications. You can also find and extract information from, and insert information into, databases.

This chapter explains how to:

- Prepare your computer and your information for publishing.
- Install and use interactive applications on your computer.
- Publish by using an Open Database Connectivity (ODBC)–compliant data source.

## Preparing Information for Publishing

Most Web pages are formatted in Hypertext Markup Language (HTML). HTML files are simple ASCII text files with codes embedded to indicate formatting and hypertext links. HTML specifications are changing constantly. You should probably review the HTML specifications (available on the Internet) to fully plan your HTML pages.

### Authoring HTML Files

You can use any text editor, such as Notepad or Write, to create and edit your HTML files; but you will probably find an HTML editor, such as Microsoft® FrontPage™ or Internet Assistant for Microsoft® Word, easier to use.

You use the HTML editor or other system to create HTML files, which can include hyperlinks to other files on your system. If you want to include images or sounds, you will also need appropriate software to create and edit those files.

### Publishing HTML and Other File Formats

Your files can include images and sound. You can even create links to Microsoft® Office files or to almost any other file format. Remote users must have the correct viewing application to view non-HTML files. For example, if you know that all remote users will

have Microsoft Word, you can include links to Microsoft Word .doc files. The user can click the link and the document will appear in Word on the user's computer.

Once you have created your information in HTML or other formats, you can either copy the information to the default directory InetPub\Wwwroot, or you can change the default home directory to the directory containing your information.

# MIME Type Configuration

If your Web site includes files that are in multiple formats, your computer must have a Multipurpose Internet Mail Extension (MIME) mapping for each file type. If MIME mapping on the server is not set up for a specific file type, browsers may not be able to retrieve the file. See the Windows NT registry for the default MIME mappings.

To configure additional MIME mappings, start the Registry Editor (Regedt32.exe) and open

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\Mim

Add a REG_SZ value for the MIME mapping required for your computer with the following syntax:

```
<mime type>,<filename extension>,<unused parameter>,<gopher type>
```

For example:

```
text/html,htm,/unused,1
image/jpeg,jpeg,/unused,5
```

The string associated with the value (that is, the value content) should be blank. The default entry with the file-name extension specified as an asterisk (*) is the default MIME type used when a MIME mapping does not exist. For example, to handle a request for the file Current.vgr when the the file-name extension .vgr is not mapped to a MIME type, your computer will use the MIME type specified for the asterisk extension, which is the type used for binary data.Usually, this will cause browsers to save the file to disk.

# Including Other Files with the Include Statement

You can add common information into HTML files just before sending the files to users. This feature is handy for including the same text on each HTML page, such as copyright information or a link to the home page.

The format of the include statement is:

```
<!--#include file="value"-->
```

The value can contain a relative path or the full path, from the home directory of your WWW service.

For example, to include a link to your home page in each HTML document:

1. Create the file Linkhome.htm, which contains the HTML codes you want to

repeat; for example, a button to your home page. The file would contain HTML code that looks similar to this:

```
<A HREF="/homepage.htm"><IMG SRC="/images/button_h.gif"></A>
```

**2.** Use the file-name extension .stm when you create your Web pages (rather than .htm or .html).

> **Notes** The .stm extension tells Peer Web Services that there is an include statement in the file. If you name the file with an .htm or .html extension, the include statement will be ignored.
>
> Using .stm files may affect performance. Therefore, use this extension only when necessary.
>
> However, in the Windows NT registry, you can change the default .stm extension to any extension you want, except for .htm or .html. For details, see the ServerSideIncludesExtension registry key in Chapter 10, "Configuring Registry Entries."

**3.** In each .stm file, use an **include file** statement where you want the repeated information to appear. For example:

```
You can return to: <!--#include file="/linkhome.htm"--> at any time
```

Note that all paths are relative to the WWW home directory and can include virtual directories.

---

# Publishing Dynamic Applications

One of the most exciting features you get when you use Microsoft Peer Web Services for Windows NT is the ability to develop applications or scripts that remote users start by clicking HTML links or by filling in and sending an HTML form. Using programming languages such as C or Perl, you can create applications or scripts that communicate with the user in dynamic HTML pages.

## Creating the Applications or Scripts

Interactive applications or scripts can be written in almost any 32-bit programming language, such as C or Perl, or as Windows NT batch files (which have the .bat or .cmd file-name extension). When you write your applications or script you can use one of two supported interfaces, Microsoft Internet Server Application Programming Interface (ISAPI) or Common Gateway Interface (CGI). Documentation for ISAPI is available from Microsoft by subscription to the Microsoft Developer Network (MSDN). You can find an introduction to CGI later in this chapter; CGI information readily accessible by way of the Internet. Batch files can issue any command valid at the command prompt.

Applications that use ISAPI are compiled as dynamic-link libraries (DLLs) that are loaded by the WWW service at startup. Because the programs are resident in memory,

ISAPI programs are significantly faster than applications written to the CGI specification.

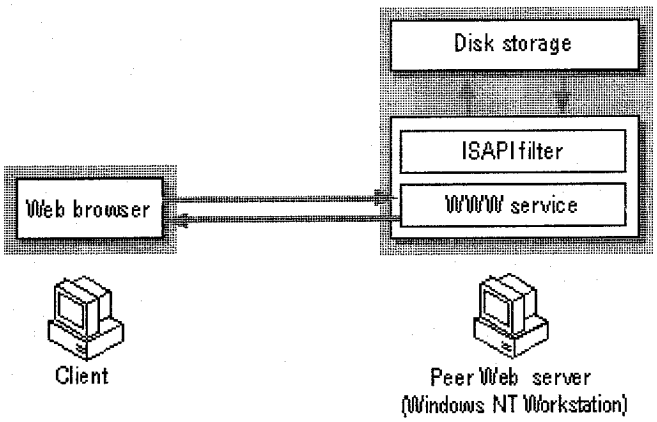
## ISAPI Perl Available for Download

Hip, Inc., the independent software vendor that develops Perl for Win32 platforms, has developed a version of Perl that runs as an ISAPI application. This means that Perl server scripts can run much faster than before by taking advantage of the in-process model of ISAPI. An unsupported release of ISAPI Perl is now available for download at http://www.perl.hip.com/. More information is available on that WWW site. Please use the perlis@mail.hip.com e-mail alias to ask questions or send feedback, especially if you have existing Perl scripts.

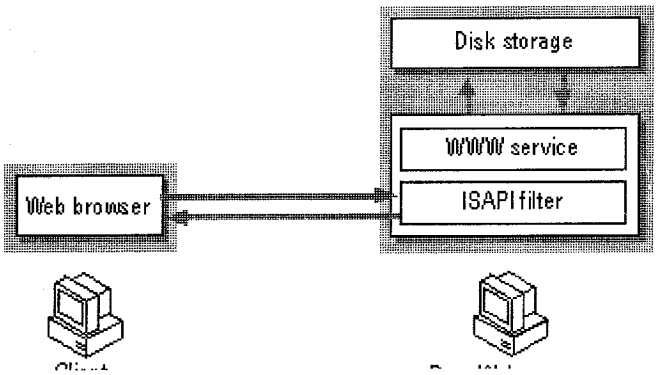
## Internet Server API

ISAPI for Windows NT can be used to write applications that Web users can activate by filling out an HTML form or clicking a link in an HTML page on your Web site. The remote application can then take the user-supplied information and do almost anything with it that can be programmed, and then return the results in an HTML page or post the information in a database.

ISAPI can be used to create applications that run as DLLs on your Web server. If you have used Common Gateway Interface (CGI) scripts before, you will find that the ISAPI applications have much better performance because your applications are loaded into memory at server run-time. They require less overhead because each request does not start a separate process.

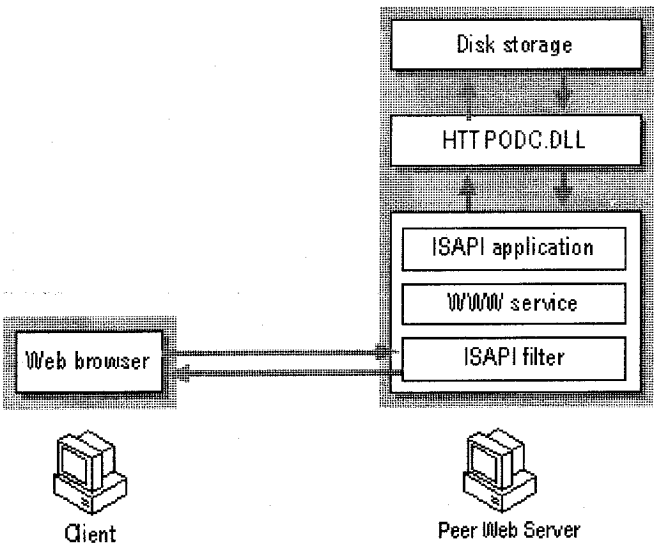



Another feature of ISAPI allows pre-processing of requests and post-processing of responses, permitting site-specific handling of Hypertext Transfer Protocol (HTTP) requests and responses. ISAPI filters can be used for applications such as customized authentication, access, or logging.



Client Peer Web server (Windows NT Workstation)

You can create very complex sites by using both ISAPI filters and applications. ISAPI extensions can also be combined with the Internet Database Connector to create highly interactive sites.



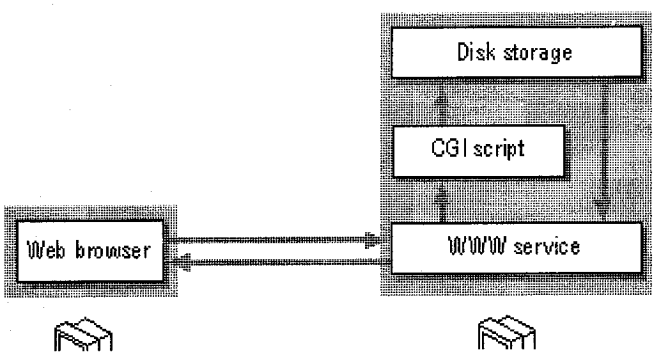

For complete information about programming with ISAPI, see the Microsoft Win32 Software Development Kit (SDK), available from MSDN. See the introductory chapter of this guide, "Before You Begin," for further information about obtaining the ISAPI SDK.

## Common Gateway Interface

Common Gateway Interface (CGI) is a set of specifications for passing information between a client Web browser, a Web server, and a CGI application. A client Web browser can start a CGI application by filling out an HTML form or clicking a link in an HTML page on your Web server. As with ISAPI, the CGI application can take the information the client Web browser supplies and do almost anything that can be programmed, then return the results of the application in an HTML page, or post the information to a database. Because simple CGI applications are often written using scripting languages such as Perl, CGI applications are sometimes referred to as "scripts."

Microsoft Peer Web Services can use most 32-bit applications that run on Windows NT and conform to the CGI specifications.

The following figure illustrates how a browser, a server, and a CGI application exchange information by using CGI. The rest of this section discusses this five-part process.





### Client Sends Request

A client browser can make a CGI request to a server by either of two methods:

GET

The client appends data to the URL it passes to the server.

POST

The client sends data to the server by way of the HTTP message data field, thereby overcoming size limitations inherent to the GET method.

The client initiates a CGI process by clicking any of the following on an HTML page:

- A hypertext link that runs the script directly.
- The "Submit" button in an HTML form.
- An inline object retrieved with the GET method.
- A search object (that is, one that uses the HTML tag ISINDEX).

### Server Receives Request

The URL that the client browser sends to the server contains the name of the CGI script or application to be run. The server compares the file name's extension to the server's Script Mapping registry key to determine which executable to launch. The server has Script Map entries for .cmd and .bat files, which launch Cmd.exe; and for .idc files, which launch the Internet Database Connector. To enable the server to launch a type of CGI application without an extension mapping, add an entry for that application type to the registry key. For example, to enable Perl scripts to run, add an entry like the following:

```
.pl: REG_SZ: C:\RESKIT\PERL\BIN\PERL.EXE %s %s
```

Where

- \Reskit\Perl\Bin\ is the directory containing the executable.
- Perl.exe is the command executed.
- the first **%s** is the translated path of the PERL script (the URL translated to a local path).
- The second **%s** is the query string (information in the URL) and is passed as a command line parameter only if the query string does not contain an equals (=)

sign.

**Server Passes Request to Application**

The server passes information to the CGI application by means of environment variables, then launches the application. Some of these variables are server-related; the majority come from the client browser and relate either to the client browser or to the request it is sending. See the table of variables at the end of this chapter for a partial list of environment variables.

**CGI Application Returns Data to Server**

The application performs its processing. If it is appropriate, the application then writes data in a format the client can receive to the standard output stream (STDOUT). The application must follow a specific format in returning data:

1. The first line or lines contain server directives, and must contain the MIME content-type. Other server directives are Location (which redirects the client to, or returns, another document) and Status.

2. A blank line must follow server directives.

3. The data the application returns to the client follows the blank line.

**Server Returns Data to Client**

The server takes the data it receives from STDOUT and adds standard HTTP headers. It then passes the HTTP message back to the client.

For more information about CGI, refer to the CGI specifications at http://hoohoo.ncsa.uiuc.edu/cgi/.

## CGI and Peer Web Services

The WWW service supports the standard Common Gateway Interface (CGI) specification. However, you should be aware of the following, unique to the implementation of CGI on Peer Web Services:

- For this release, only 32-bit CGI applications work with the WWW service.
- The REMOTE_USER environment variable is not present when the user is logged in as the anonymous user (that is, when the Web server is accessed anonymously).
- All of the variables defined for ISAPI applications are passed to the CGI application as environment variables.

Note that CGI applications are typically stand-alone executables. This is in contrast to ISAPI applications, which are typically loaded as DLLs and are therefore server extensions. Thus, ISAPI applications offer enhanced performance when compared to CGI applications and scripts.

### Security Considerations for Executables

Common Gateway Interface (CGI) executables must be used with extreme caution to prevent potential security risk to the server. As a rule, give only Execute permission to virtual directories that contain CGI or Internet Server API (ISAPI) applications.

It is highly recommended that you configure script mapping. Script mapping ensures that the correct interpreter (Cmd.exe, for example) starts when a client requests an executable file.

World Wide Web content directories should be assigned Read permission only. Any executable files intended for downloading from Windows NT File System (NTFS) drives should have only Read access enabled.

You can run batch files as CGI executable files, but you must do so with extreme caution to prevent potential security risk to the server.

**Note** CGI executable files can also have the file-name extension .exe or .cgi.

### Execute Permission for ISAPI Applications

Peer Web Services opens ISAPI applications in the security context of the calling user. An access check is performed against that calling user. To restrict execution to selected users, NTFS permissions can be used with ISAPI applications such as the Internet Database Connector (IDC).

For example, to secure the IDC without putting permissions on the .idc file, you can grant NTFS Execute permission for Inetsrv\Httpodbc.dll to the appropriate users. Httpodbc.dll is the name of the ISAPI application DLL that implements the IDC. Then, whenever a user tries to execute the IDC, the server will check the permissions. Access will be allowed only if Execute permission has been granted for that user.

**Note** Once an ISAPI application has been loaded, it remains loaded until the WWW service is stopped. Peer Web Services does not track security descriptor changes after the ISAPI application has been loaded. If you change permissions for an ISAPI application after it has been loaded, you must stop and restart the WWW service before the change will take effect.

Take care in setting Access Control Lists (ACLs) on the Winnt directory and its subdirectories. Some ISAPI applications and databases require access to files and DLLs in these directories.

**Note** ISAPI application DLLs can have the file-name extension .dll or .isa.

## Installing Your Application on Peer Web Services

Once you have written your application or script, place it in the Scripts directory, a virtual directory for applications. This virtual directory has Execute access.

You must also ensure that every process started by your application is running by using an account with adequate permissions. If your application interacts with other files, the

account you assign to your program must have the appropriate permissions to use those files. By default, applications run using the IUSR_*computername* account, which must have adminstrator and execute permissions for these application files.

# Running Your Application

If your application does not require data from the user, you will typically create a link to your application in a simple HTML file. If your application does require data from the user, you will probably use an HTML form. In other instances you can just send a Uniform Resource Locator (URL), usually containing data parameters, to invoke a program.

An HTML link to an application that does not require input from the user might look like the following example:

```
http://www.company.com/scripts/catalog.exe
```

where Scripts is the virtual directory for interactive applications.

If you are creating an application that requires input from the user, you will need to understand both HTML forms and how to use the forms with ISAPI or CGI. This information is widely available on the Internet or from other sources.

## Associating Interpreters with Applications

Because you have the flexibility to create applications in almost any programming language, Peer Web Services uses the file-name extension to determine which interpreter to invoke for each application. The default interpreter associations are listed below. You can use the Registry Editor to create additional associations.

| Extension | Default Interpreter |
|---|---|
| .bat, .cmd | Cmd.exe |
| .idc | Httpodbc.dll |
| .exe, .com | System |

## Security Implications

When you allow remote users to run applications on your computer, you run the risk of hackers attempting to break into your system. Microsoft Peer Web Services is configured by default to reduce the risk of malicious intrusion by applications in two important ways.

First, the virtual directory Scripts contains your applications. Only an administrator can add programs to a directory marked as an execute-only directory. Thus, unauthorized users cannot copy a malicious application and then run it on your computer without first gaining administrator access.

It is recommended that you grant read and execute permission for the IUSR_*computername* account on the directory associated with the virtual folder, and full control only to the administrator. Perl scripts (.pl file-name extension) and IDC files (.idc

and .htx file-name extensions) need both read and execute permission. However, to prevent someone from installing an unsafe file on your server, do not grant write permission.

Second, if you have configured the WWW service to allow only anonymous logons, all requests from remote users will use the IUSR_*computername* account. By default, the IUSR_*computername* account is unable to delete or change files by using the Windows NT File System (NTFS) unless specifically granted access by an administrator. Thus, even if a malicious program were copied to your computer, it would be unable to cause much damage to your content because it will only have IUSR_*computername* access to your computer and files.

---

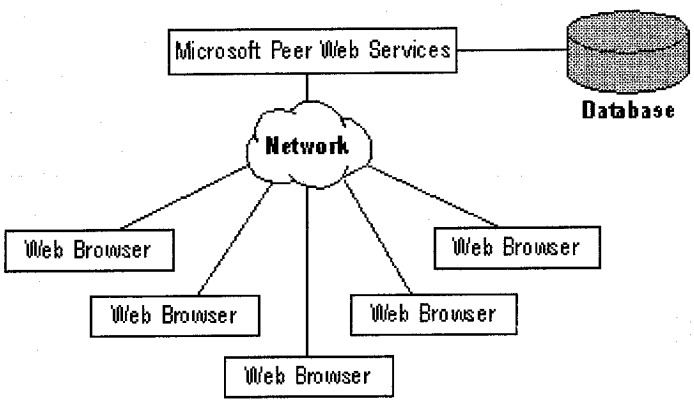
# Publishing Information and Using a Database

With the WWW service and the Open Database Connectivity (ODBC) drivers provided with Peer Web Services, you can:

- Create Web pages with information contained in a database.
- Insert, update, and delete information in the database based on user input from a Web page.
- Perform other Structured Query Language (SQL) commands.

## How the Internet Database Connector Works

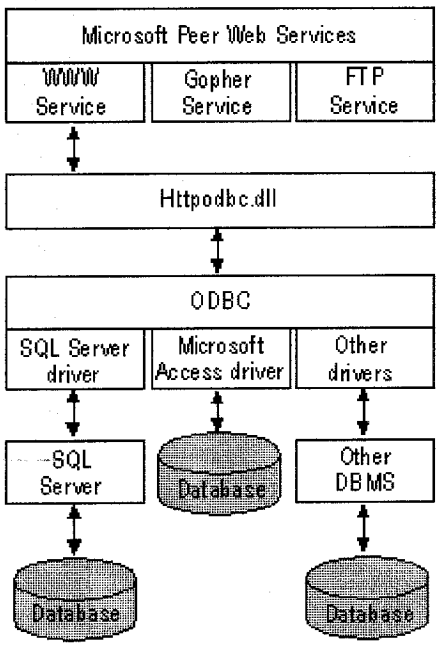
Conceptually, database access is performed by Peer Web Services as shown in the following diagram.




Web browsers (such as Internet Explorer, or browsers from other companies such as Netscape) submit requests to the Internet server by using HTTP. The Internet server responds with a document formatted in HTML. Access to databases is accomplished through a component of Peer Web Services called the Internet Database Connector (IDC). The Internet Database Connector, Httpodbc.dll, is an ISAPI DLL that uses ODBC to gain access to databases.

The following illustration shows the components for connecting to databases from Peer

Web Services.




The IDC uses two types of files to control how the database is accessed and how the output Web page is constructed. These files are Internet Database Connector (.idc) files and HTML extension (.htx) files.

The Internet Database Connector files contain the necessary information to connect to the appropriate ODBC data source and execute the SQL statement. An Internet Database Connector file also contains the name and location of the HTML extension file.

The HTML extension file is the template for the actual HTML document that will be returned to the Web browser after the database information has been merged into it by the IDC.

# Installing ODBC and Creating System Data Sources

When the ODBC option is selected during setup, ODBC version 2.5 components are installed. This version of ODBC supports System DSNs (Data Source Names) and is required for using ODBC with Microsoft Peer Web Services.

System DSNs were introduced in ODBC version 2.5 to allow Windows NT services to use ODBC.

**To install the ODBC drivers**

1. If you did not install the ODBC Drivers and Administration option, run Setup again by clicking the Peer Web Services Setup icon in the Microsoft Peer Web Services program group. You will need the Windows NT Workstation compact disc, or a network installation directory containing the complete contents of the compact disc.

2. Click the **OK** button.

3. Click the **Add/Remove** button.

4. Click the **OK** button.

5. Select the **ODBC Drivers and Administration** option.

6. Click the **OK** button.

7. The **Install Drivers** dialog box appears.

8. To install the SQL Server driver, select the SQL Server driver from the **Available ODBC Drivers** list box, and click the **OK** button.

   Setup completes copying files.

**To create the system data sources**

1. Double-click the Control Panel icon in the Main program group of Program Manager.

2. Double-click the ODBC icon.

   The **ODBC Data Sources** dialog box appears.

   You may see other data sources in the list if you previously installed other ODBC drivers.

3. Click the **System DSN** button.

---

**Important** Be sure to click the **System DSN** button. The Internet Database Connector will work only with System DSNs.

---

   The **System Data Sources** dialog box appears.

4. Click the **Add** button.

   The **Add Data Source** dialog box appears.

5. Select an ODBC driver in the list box and click **OK**. A dialog box specific to your driver will appear.

6. Enter the name of the data source.

   The data source name is a logical name used by ODBC to refer to the driver and any other information required to access the data, such as the actual server name or location of the database. The data source name is used in Internet Database Connector files to tell Peer Web Services where to access the data.

   For Microsoft SQL Server, the server name, network address, and network library displayed in the Setup dialog box are specific to your installation. If you do not know what to enter in these controls, accept the defaults. To find out the details, click the **Help** button and find the section that describes your

network.

7. Click the **OK** button.

   The **System Data Sources** dialog box will be displayed again, but now will have the name of the data source displayed.

8. Click the **Close** button to close the **System Data Sources** dialog box.

9. Click the **Close** button to close the **Data Sources** dialog box.

10. Click the **OK** button to complete the ODBC and DSN setup.

### 32-Bit ODBC Drivers

The Internet Database Connector requires 32-bit ODBC drivers. Refer to the Peer Web Services Help files or the Windows NT ODBC Help file for information about the ODBC option.

### Microsoft Access ODBC Drivers

The Internet Database Connector requires the 32-bit ODBC drivers shipped with Microsoft® Office 95 and Microsoft® Access 95. The ODBC driver for Microsoft Access 2.0 will not work with Peer Web Services.

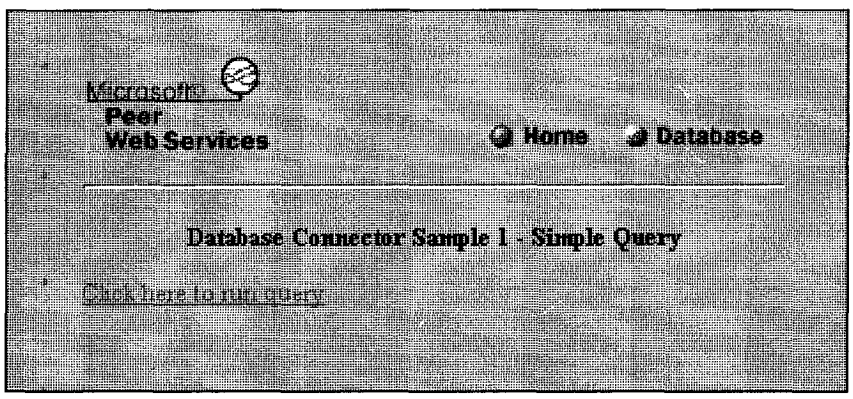
## Authoring Web Pages with Database Access

In order to provide access to a SQL database from your Web page, you will need to create an Internet Database Connector file (.idc file-name extension) and an HTML extension file (.htx file-name extension).

## Walking through a Sample Database Query

This example starts with a simple Web page called Sample.htm. The sample Web page will contain one hyperlink that will result in a query being executed using the ODBC driver for Microsoft SQL Server, with the results returned as another Web page.

The following graphic shows Dbsamp1.htm as it is displayed by Microsoft Internet Explorer (assuming that Peer Web Services has been installed on a computer called "webserver").

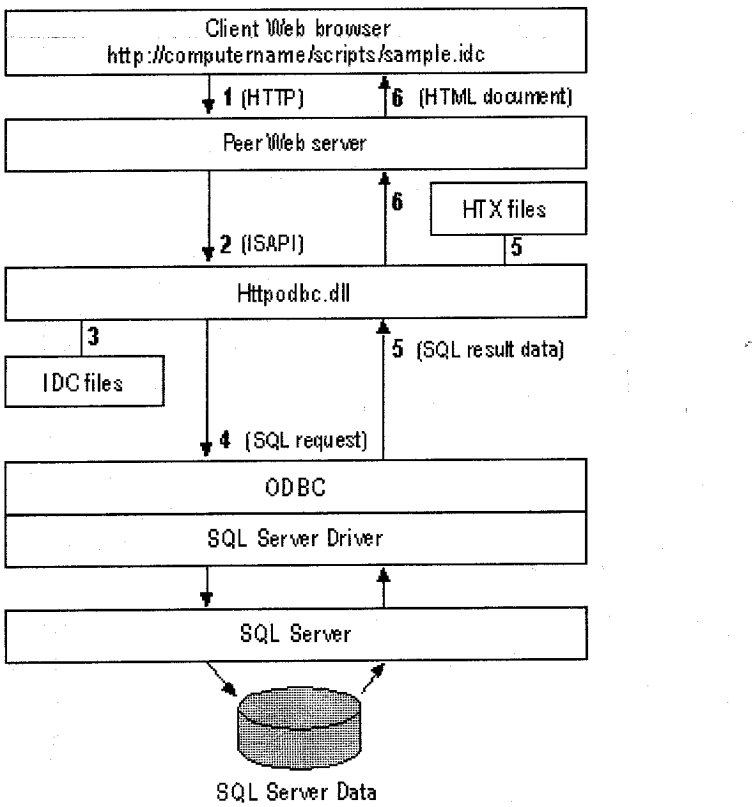


When the hyperlink "Click here to run query" is clicked, another Uniform Resource Locator (URL) is sent to the server. The URL precedes the hyperlink text (it is formatted as hidden text):

```
<A HREF="http://webserver/samples/dbsamp/dbsamp1.idc">Click here to run
query</A>
```

The Internet Database Connector file for the IDC to use (Dbsamp1.idc) is referenced in the URL. Extension file mapping precludes the need to reference Httpodbc.dll in the URL.

On Peer Web Services, the entire process of using the Internet Database Connector for this example is performed in six steps, as shown in the following diagram.




**1. The URL is received by Peer Web Services.**

The URL is sent by the Web browser.

**2. Peer Web Services loads Httpodbc.dll and provides it with the remaining information in the URL.**

.Idc files are mapped to Httpodbc.dll. Httpodbc.dll loads and obtains the name of the Internet Database Connector file (and other items) from the URL passed to Peer Web Services.

**3. Httpodbc.dll reads the Internet Database Connector file.**

The Internet Database Connector file contains several entries in the format

```
field: value
```

In the Sample.idc file, the ODBC data source is specified by:

```
Datasource: Web SQL
```

And the HTML extension file is specified by:

```
Template: sample.htx
```

Here are the entire contents of the file Sample.idc referenced in the preceding URL:

```
Datasource: Web SQL
Username: sa
Template: sample.htx
SQLStatement:
+SELECT au_lname, ytd_sales
+ from pubs.dbo.titleview
+ where ytd_sales>5000
```

In the sample .idc file the data source name is "Web SQL" The ODBC installation notes tell how to create a data source called Web SQL.

The other items contained in the sample .idc file include:

- User name, which must be a valid logon to the ODBC datasource; in this example, the logon is to the "sa" account on a Microsoft SQL Server.
- Template, which specifies the file to use to merge the results.
- SQL Statement, which contains the SQL statement to execute.

See "Learning the Features of the Internet Database Connector," later in this chapter, for definitions for all the fields that can be specified in Internet Database Connector files.

The SQLStatement in Sample.idc returns all the author last names and year-to-date sales in units from the "pubs" sample database in SQL Server for authors whose books have year-to-date sales of more than 5000 dollars.

**4. The IDC connects to the ODBC data source, and executes the SQL statement contained within the Internet Database Connector file.**

The connection is made to the ODBC data source by the IDC, which in this example loads the ODBC driver for SQL Server and connects to the server specified in the definition of the data source. Once the connection is made, the SQLStatement in the Internet Database Connector file is sent to the SQL Server ODBC driver, which in turn sends it to SQL Server.

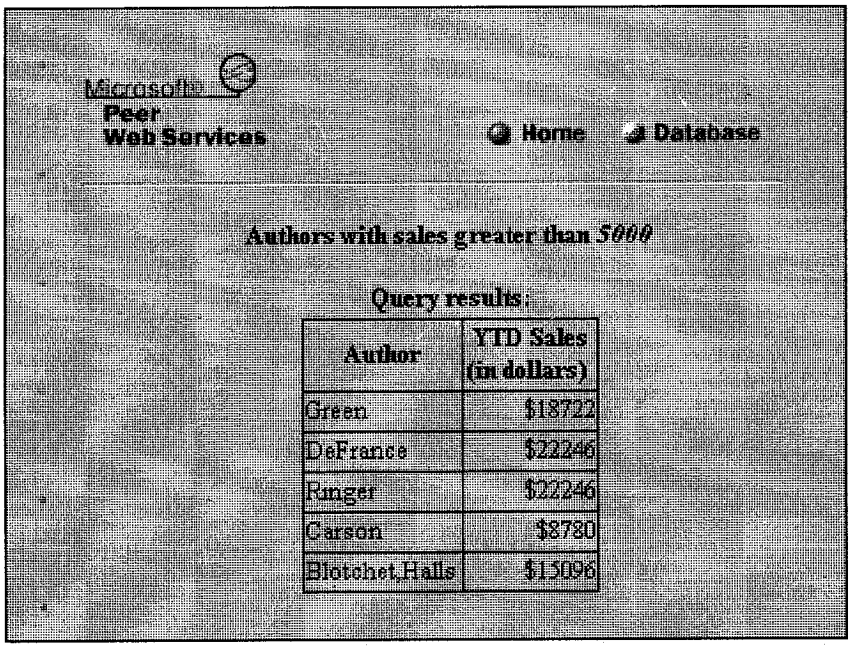
**5. The IDC fetches the data from the database, and merges it into the HTML extension file.**

After the SQL statement has been executed, IDC reads the HTML extension file specified in Sample.idc (Sample.htx). HTML extension (.htx) files contains special HTML tags which IDC uses to control where and how the data returned from the SQL statement is merged.

**6. The IDC sends the merged document back to Peer Web Services, which returns it to the client.**

After all the data has been merged into Sample.htx, the complete HTML document is sent back to the client.

The resulting Web page is displayed in the Microsoft Internet Explorer as shown following.




# Understanding the Sample.htx File

To return data to the WWW client, the .idc file merges the HTML extension .htx file and the ODBC data. This combined data is attached to standard HTTP headers (200 OK status, Content-Type, and so on) and passed to the WWW service and returned to the client.

The .htx file is an HTML document with some additional tags enclosed by <%%> or <!--%%-->, which the .idc file uses to add dynamic data to the document. The HTML formatting in the .htx file typically formats the data being returned. There are six keywords (begindetail, enddetail, if, else, endif, and "%z") that control how the data from the database is merged with the HTML format in the .htx file. Database column names specify what data is returned in the HTML document. For example, the following line in an .htx file merges data from the Emailname column for every record processed:

```
<%begindetail%><%Emailname%><%enddetail%>
```

The Sample.htx file is an HTML document that contains Internet Database Connector tags for data returned from the database (the tags are shown in bold for clarity). Some

HTML formatting has been removed to highlight the IDC tags.

Most of the HTML formatting has been removed for clarity.

```
<HTML>
<BODY>
<HEAD><TITLE>Authors and YTD Sales</TITLE></HEAD>
<%if idc.sales eq ""%>

<H2>Authors with sales greater than <I>5000</I></H2>
<%else%>

<H2>Authors with sales greater than <I><%idc.sales%></I></H2>
<%endif%>

<P>
<%begindetail%>
<%if CurrentRecord EQ 0 %>

Query results:
<B>Author YTD Sales<BR></B>
<%endif%>
<%au_lname%><%ytd_sales%>
<%enddetail%>
<P>
<%if CurrentRecord EQ 0 %>
<I><B>Sorry, no authors had YTD sales greater than </I><%idc.sales%>.</B>

<P>
<%else%>

<HR>
<I>
The Web page you see here was created by merging the results of the SQL query with
the template file Sample.htx.
<P>
The merge was done by the Microsoft Internet Database Connector and the results were
returned to this Web browser by the Microsoft Peer Web Services.
</I>
<%endif%>

</BODY>
</HTML>
```

The <%begindetail%> and <%enddetail%> sections delimit where rows returned from the database will appear in the document. Columns returned from the query are surrounded by <%%>, such as <%au_lname%> and <%ytd_sales%> in this example.

# Learning the Features of the Internet Database Connector

The Internet Database Connector has several features that help create Web pages

containing data from a database.

## Internet Database Connector Files

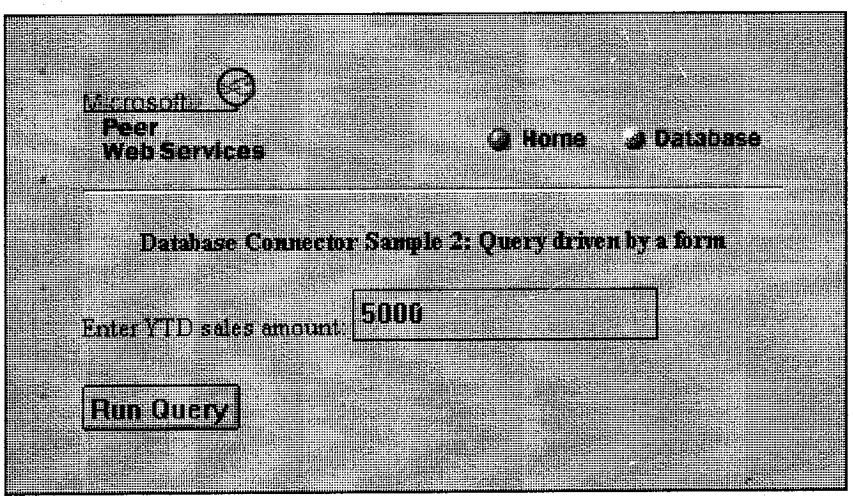
Internet Database Connector files contain the information used to access the database. The following section describes the features of Internet Database Connector files.

### Parameters

The example in the preceding section shows only the simplest kind of query, a query that was defined completely in the Internet Database Connector file. Although this type of query is useful, even more powerful Web pages can be constructed through the use of parameters. Parameters are the names and values of HTML-form controls, such as "<INPUT...>", and names specified directly in URLs. These names and values are sent by Web browsers and can be used in SQL statements on the server.

For example, in the last section the query in Sample.idc returned only the authors whose year-to-date sales exceeded 5000. By using a parameter, you could build a Web page that asks the user to decide what number to use instead of 5000.

The Web page must prompt the user for the year-to-date sales figure and then name the associated variable to "sales." Dbsamp2.htm shows a form with an input field used to obtain the number:




The HTML syntax for the input field and button in Sample2.htm is:

```
<FORM METHOD="POST" ACTION="/scripts/samples/sample2.idc">
<P>
Enter YTD sales amount: <INPUT NAME="sales" VALUE="5000" >
<P>
<INPUT TYPE="SUBMIT" VALUE="Run Query">
</FORM>
```

In the Internet Database Connector file Sample2.idc, you use the parameter shown in bold in place of the number 5000:

```
SQLStatement:
+SELECT au_lname, ytd_sales
+ from pubs.dbo.titleview
+ where ytd_sales > %sales%
```

Here the parameter name must be "sales" so that it corresponds to the <INPUT NAME= "sales" ...> on the Web page. Parameters must be enclosed with percent characters (%) to distinguish them from a normal identifier in SQL. When the Internet Database Connector encounters the parameter in the .idc file, the Internet Database Connector substitutes the value sent by the Web browser and then sends the SQL statement to the ODBC driver.

The percent character (%) is also a wildcard character in SQL. You can use wildcards in an SQL query to search for an element in a table that contains certain characters. To insert a single "%" for a SQL wildcard, use "%%." This prevents the IDC from trying to use the % as a parameter marker. For example:

```
SQLStatement:
+SELECT au_lname, ytd_sales, title
+ from pubs.dbo.titleview
+ where title like '%%%title%%%'
```

For a percent sign to be recognized as an SQL wildcard you must double it and then add the percent characters around the parameter to distinguish the string as a parameter. In the example, the query searches for all entries in the title column with the word *title* in them. This query returns the following:

```
title
title and deed
main title page
author and title
```

To return all entries with the word *title* as the first five letters, you would format the query as follows:

```
SQLStatement:
+SELECT au_lname, ytd_sales, title
+ from pubs.dbo.titleview
+ where title like '%title%%%'
```

In this example, the following results are returned:

```
title
title and deed
```

To return all entries with the word *title* as the last five letters, you format the query as follows:

```
SQLStatement:
+SELECT au_lname, ytd_sales, title
+ from pubs.dbo.titleview
+ where title like '%%%title%'
```

In this example, the following results are returned:

```
title
author and title
```

You can build powerful collections of Web pages by using the output of one query to provide links to other queries. For example, to show the titles for an individual author,

instead of returning the author name as plain text, you can format it as a link and then use the link to do another query.

Another example included Peer Web Services shows how to do this type of linkage. Dbsamp3.htm is used to run the query in Sample3.idc, which uses Sample3.htx for the output template. Sample3.htx will return author last names as links, which, when clicked, will display the titles for each author by using Sample3a.idc and Sample3a.htx.

### Fields in Internet Database Connector (.idc) Files

The following tables list the fields that can be specified in an Internet Database Connector file. Note that parameters or server variables may appear anywhere in an .idc file.

**Required Fields in an Internet Database Connector (.idc) File**

| Field | Description |
|---|---|
| Datasource | The name that corresponds to the ODBC system Data Source Name (DSN) you created earlier by using the ODBC Administrator or the tool provided with the samples. |
| Template | The name of the HTML extension file that formats the data returned from this query. By convention these files use the file-name extension .htx. |
| SQLStatement | The SQL statement to execute. The SQL statement can contain parameter values, which must be enclosed with percent characters (%) from the client. The SQLStatement can occupy multiple lines in the Internet Database Connector file. Following the SQLStatement field, each subsequent line beginning with a plus sign (+) is considered part of the SQLStatement field. Multiple SQLStatements can appear in the same file. |

**Optional Fields in an Internet Database Connector (.idc) File**

| Field | Description |
|---|---|
| DefaultParameters = *param=value*<br><br>[, *param=value*]<br>[...] | The parameter values, if any, that will be used in the Internet Database Connector file if a parameter is not specified by the client. |
| Expires | The number of seconds to wait before refreshing a cached output page. If a subsequent request is identical, the cached page will be returned without ever accessing the database. The Expires field is useful when you want to force a requery of the database after a certain period of time. The IDC does not cache output pages by default. It caches them only when the Expires field is used. |
| MaxFieldSize | The maximum buffer space allocated by the IDC per field. Any characters beyond this will be truncated. The parameter applies only to fields returned from the database that exceed 8192 bytes. The default value is 8192 bytes. |
| MaxRecords | The maximum number of records that the IDC will return from any one query. The MaxRecords value is not set by default, meaning that a query can return up to 4 billion records. Set this value to limit the records returned. |

| | |
|---|---|
| ODBCConnection | Insert this field with the value of *pool* to add the connection to the connection pool, which keeps the connection to the database open for future requests. The IDC then sends data through a pooled connection for subsequent execution of an .idc file that contains the same values for Datasource, Username, and Password. Set this option to improve performance using the Internet Database Connector. Also, there is a *nonpool* option, which specifies that the connection for the .idc file in which this option is set should not be taken from the connection pool. Set the value of this field to **nopool** to manage the cache of connections more precisely. Also, if there is a limit on the number of current connections, you do not want the connection pool to monopolize all the connections; otherwise, no one else could connect to the SQL Server.<br><br>**Note** To set the default to connection pooling, you must set the PoolIDCConnections registry entry to 1. For details, see Chapter 10, "Configuring Registry Entries." |
| Password | The password that corresponds to the user name. If the password is null, this field can be left out. |
| RequiredParameters | The parameter names, if any, that Httpodbc.dll will ensure are passed from the client; otherwise, it will return an error. Parameter names are separated with a comma. |
| Translationfile | The path to the file that maps non-English characters (such as à, ô, or é) so that browsers can display them properly in HTML format. If the translation file is not in the same directory as the .idc file, you must type the full path to the translation file. Syntax: Translationfile: C:*directoryname**filename*. Use the Translationfile field if you are publishing a database in a language other than English. A translation file is a text file with each special character mapped in the following format: *value*=*string*<CR> where *value* is an international character and *string* is the HTML translation code. |
| Username | A valid user name for the data source name supplied in the Datasource field.<br><br>**Note** If you use Microsoft SQL Server with the integrated security option, the username and password fields in the .idc file are ignored. The logon to SQL Server is performed using the credentials of the Web user. If the request is made as the anonymous user, the username and password are determined by the settings for the anonymous user (IUSR_*computername* by default) in the Internet Service Manager. If the client request contained logon credentials, the username and password supplied by the end user are used to log on to SQL Server. |
| Content-Type | Any valid MIME type describing what will be returned to the client. Almost always this will be "text/html" if the .htx file contains HTML. |

## ODBC Advanced Optional Fields

ODBC advanced options allow debugging and fine-tuning of the ODBC driver used by the Internet Database Connector. For more details about these options, consult your ODBC driver documentation or the ODBC Software Development Kit (SDK). The format in the IDC file is:

```
ODBCOptions: Option Name=Value[,Option Name=Value...]
```

For example, to stop the SQL statement from running for more than 10 seconds and enabling tracing of ODBC function calls, in the IDC file you would specify:

```
ODBCOptions: SQL_QUERY_TIMEOUT=10, SQL_OPT_TRACE=1,
SQL_OPT_TRACEFILE=C:\Sql.log
```

All options are described in the following table:

| Option Name | Value | Purpose |
|---|---|---|
| SQL_ACCESS_MODE | 0 = Read/Write<br><br>1 = Read Only. | An indicator for the ODBC driver or data source that the connection is not required to support SQL statements that cause updates to occur. This mode can be used to optimize locking strategies, transaction management, or other areas as appropriate to the driver or data source. The driver is not required to prevent such statements from being submitted to the data source. The behavior of the driver and data source when asked to process SQL statements that are not read-only during a read-only connection is implementation-defined. SQL_ACCESS_MODE set to 0 is the default, which allows reading and writing. |
| SQL_LOGIN_TIMEOUT | Integer | The number of seconds to wait for a logon request to complete before disconnecting. The default is driver-dependent and must be nonzero. If the value is 0, the timeout is disabled and a connection attempt will wait indefinitely. If the specified timeout exceeds the maximum log on timeout in the data source, the driver substitutes that value. |
| SQL_OPT_TRACE | 0 = Trace off<br><br>1 = Trace on | When tracing is on, each ODBC function call made by Httpodbc.dll is written to the trace file. You can specify a trace file with the SQL_OPT_TRACEFILE option. If the file already exists, the ODBC appends to the file. Otherwise, it creates the file. If tracing is on and no trace file has been specified, ODBC writes to the file Sql.log. |
| SQL_OPT_TRACEFILE | File name | The name of the trace file to use when SQL_OPT_TRACE=1. The default is SQL.LOG |
| SQL_PACKET_SIZE | Integer | The network packet size, in bytes, to be used to exchange information between the database management system (DBMS) and the Web Server.<br><br>**Note** Many data sources either do not support this option or can return only the network packet size. If the specified size exceeds the maximum packet size or is smaller than the minimum packet size, the driver substitutes that value. |
| SQL_TRANSLATE_DLL | File name | The name of a DLL containing the functions SQLDriverToDataSource and SQLDataSourceToDriver that the driver loads and uses to perform tasks such as character-set translation. |

| | | |
|---|---|---|
| SQL_TRANSLATE_OPTION | Integer | Value controlling translation functionality, which is specific to the translation DLL being used. Consult the documentation for the driver and translation DLL for details. |
| SQL_TXN_ISOLATION | Integer<br><br>1=Read Uncommited<br><br>2=Read Committed<br><br>4=Repeatable Read<br><br>8=Serializable<br><br>16=Versioning | Sets the transaction isolation level. The Internet Database Connector does not support transactions than span more than the request in the .idc file. However, for some DBMSs, setting the SQL_TXN_ISOLATION option to 1 (Read Uncommited) will result in higher concurrency and therefore better performance. However, with this setting, data that has not been committed to the database by other transactions may be retrieved . |
| SQL_MAX_LENGTH | Integer | The maximum amount of data that the driver returns from a character or binary column. This option is intended to reduce network traffic and should only be used when the data source (as opposed to the driver) in a multiple-tier driver can implement it. |
| SQL_MAX_ROWS | Integer | The maximum number of rows to return for a SELECT statement. If the value equals 0 (the default), then the driver returns all rows. This option is intended to reduce network traffic when the data source itself can limit the return rows, as opposed to the MaxRecords built-in variable in the Internet Database Connector, which limits the rows fetched. |
| SQL_NOSCAN | 0=Scan for and convert escape clauses<br><br>1=Do not scan for and convert escape clauses | Specifies whether the driver does not scan SQL strings for escape clauses. If set to 0 (the default), the driver scans SQL strings for escape clauses. If set to 1, the driver does not scan SQL strings for escape clauses; instead, the driver sends the statement directly to the data source. If your SQL statement does not contain any ODBC escape clauses, a special syntax enclosed by curly braces ( { } ), then setting this option to 1 will provide a small performance gain by directing the driver to not scan the SQL string. |
| SQL_QUERY_TIMEOUT | Integer<br><br>0=No timeout | The number of seconds to wait for a SQL statement to execute before canceling the query. When set to 0 (the default) there is no timeout. If the specified timeout exceeds the maximum timeout in the data source, or is smaller than the minimum timeout, the driver substitutes that value. |

| | | |
|---|---|---|
| Integer | Driver Specific | Driver-specific option values can be specified in the form *number=value*. For example,<br><br>`4322=1, 234=String` |

## Using Select Multiple List Boxes in HTML Forms

When an HTML form containing a <SELECT MULTIPLE...> tag is used, the Internet Database Connector converts the items selected into a comma-separated list; the list can be used in the .idc file just like other parameters. However, because the parameter is actually a list, it will typically only be used for SQLSelect statements with an IN clause, as in the following examples.

If the parameter name in the .idc file is enclosed in single quotation marks, each element of the list will be enclosed in single quotation marks also. You should enclose the parameter name in single quotation marks whenever the column in the IN clause is a character column or other type in which literals are quoted (dates and times, for example). If there are no single quotation marks around the parameter name, no quotation marks will be placed around each element of the list. You should not enclose the parameter name in single quotation marks when the column in the IN clause is a numeric type or any other type in which literals are not enclosed in single quotation marks.

For example, if an HTML form contained the multiple-choice list box shown below:

```
<SELECT MULTIPLE NAME="region">
<OPTION VALUE="Western">
<OPTION VALUE="Eastern">
<OPTION VALUE="Northern">
<OPTION VALUE="Southern">
</SELECT>
```

You can construct an .idc file with an SQL statement:

```
SQLStatement: SELECT name, region FROM customer WHERE region IN ('%region%')
```

If the user selected "Northern," "Western," and "Eastern" from the HTML form, the SQL statement would be converted to:

```
SELECT name, region FROM customer WHERE region IN ('Northern', 'Western',
'Eastern')
```

Another example of an HTML form is shown below, but this time uses numeric data, and therefore no quotation marks enclose the parameter in the .idc file.

```
<SELECT MULTIPLE NAME="year">
<OPTION VALUE="1994">
<OPTION VALUE="1995">
<OPTION VALUE="1996">
</SELECT>
```

You can construct an .idc file with an SQLStatement:

```
SQLStatement: SELECT product, sales_year FROM sales WHERE sales_year IN (%
```

```
year%)
```

If the user selected "1994" and "1995" from the HTML form, the SQL statement would be converted to:

```
SELECT product, sales_year FROM sales WHERE sales_year IN (1994, 1995)
```

## Using Batch Queries and Multiple Queries

In an .idc file, you can group SQL queries in two ways, as batch queries or as multiple queries.

### Batch Queries

If you are querying databases that can simultaneously process several queries in a SQL statement (such as SQL Server database), you should format your statements in batch query syntax to optimize performance. For example:

```
SQLSTatement:
+insert into perf(testtime, tag) values (getdate(), '%tag%')
+SELECT au_lname, ytd_sales from pubs.dbo.titleview where ytd_sales>5000
+SELECT count(*) as nrecs from pubs.dbo.titleview where ytd_sales>5000
```

### Multiple Queries

If you are querying databases that cannot process a series of SQL queries simultaneously, then formulate your queries as multiple queries. For example:

```
SQLStatement:
+insert into perf(testtime, tag) values (getdate(), '%tag%')
SQLStatement:
+SELECT au_lname, ytd_sales from pubs.dbo.titleview where ytd_sales>5000
SQLStatement:
+SELECT count(*) as nrecs from pubs.dbo.titleview where ytd_sales>5000
```

Batch queries are processed together at once, whereas multiple queries are processed one at a time. Therefore, you will get better performance by formatting your queries as a batch if your database can handle batch queries.

## HTML Extension (.htx) Files

HTML extension files contain a number of keywords that control how the output HTML document is constructed. These keywords are explained in the following sections.

### <%begindetail%>, <%enddetail%>

The <%begindetail%>, <%enddetail%> keywords surround a section of the HTML extension file in which the data output from the database will be merged. Within the section, the column names delimited with <% and %> or <!--%%-->are used to mark the position of the returned data from the query. For example:

```
<%begindetail%>
<%au_lname%>: <%ytd_sales%>
<%enddetail%>
```

will list the columns au_lname and ytd_sales. Any column can be referred to in this way. Column names can also be referred to elsewhere in the HTML extension file.

**Note** If there are no records returned from the query, the <%begindetail%> section will be skipped. For each SQL statement that generates a result set (for example, SELECT), there should be a corresponding <%begindetail%> <%enddetail%> section in the .htx file.

### <%if%>, <%else%>, <%endif%>

HTML extension files can contain conditional logic with an if-then-else statement to control how the Web page is constructed. For example, one common usage is to insert a condition to display the results from the query on the first row within a <%begindetail%> section; but if there are no records returned by the query, to display the text "Sorry, no authors had YTD sales greater than" **%idc.sales%**. By using the <%if%> statement and a built-in variable called "CurrentRecord" you can tailor the output so that the error message is printed when no records are returned. Here is an example showing the use of the <%if%> statement.

```
<%begindetail%><%if CurrentRecord EQ 0 %>

Query results:

<B>Author YTD Sales<BR></B>
<%endif%>
<%au_lname%><%ytd_sales%>
<%enddetail%>
<P>
<%if CurrentRecord EQ 0 %>
<I><B>Sorry, no authors had YTD sales greater than </I><%idc.sales%>.</B>
<P>
<%else%>
<HR>
<I>
The Web page you see here was created by merging the results of the SQL query
with the template file Sample.htx.
<P>
The merge was done by the Microsoft Internet Database Connector and the
results were returned to this Web browser by the Microsoft Peer Web Services.
</I>
<%endif%>

</BODY>
</HTML>
```

The general syntax is:

```
<%if condition %>
HTML text
[<%else%>
HTML text]
<%endif%>
```

Where *condition* is of the form:

*value1 operator value2*

and *operator* can be one of the following:

| | |
|---|---|
| EQ | if *value1* equals *value2* |
| LT | if *value1* is less than *value2* |
| GT | if *value1* is greater than *value2* |
| CONTAINS | if any part of *value1* contains the string *value2* |

The operands *value1* and *value2* can be column names, one of the built-in variables (CurrentRecord or MaxRecords, see below), an HTTP variable name (see following), or a constant. When used in an <%if%> statement, values are not delimited with <% and %>. For example, to do special processing on author name "Green," use the condition:

```
<%begindetail%>
<%if au_lname EQ "Green"%>
this guy is green!
<%endif%>
<%enddetail%>
```

The <%if%> statement can also be used to do special processing based on information from HTTP variables. For example, to format a page differently based on the type of client Web browser you could include the following in the HTML extension file.

```
<%if HTTP_USER_AGENT contains "Mozilla"%>
client supports advanced HTML features
<%else%>
client is <%HTTP_USER_AGENT%>
<%endif%>
```

### CurrentRecord, MaxRecords

The CurrentRecord built-in variable contains the number of times the <%begindetail%> section has been processed. The first time through the <%begindetail%> section, the value is zero. Subsequently, the value of CurrentRecord changes every time another record is fetched from the database.

The MaxRecords built-in variable contains the value of the MaxRecords field in the Internet Database Connector file. MaxRecords and CurrentRecord can only be used in <%if%> statements.

### Parameters from Internet Database Connector files

Parameters from Internet Database Connector files can be accessed in the HTML extension file by prefixing the name of the parameter with "idc" and a period. In Sample3.htx shown earlier, you could show the value of the parameter %sales% by including the line:

```
The value of the sales parameter is: <%idc.sales%>
```

### HTTP variables

Several variables in HTML extension files can give a lot of information about the environment and Web client connected to the server. In addition all headers sent by the

client are available. To access them by using the Internet Database Connector you must convert them:

1. Add HTTP_ to the beginning.
2. Convert all dashes to underscores.
3. Convert all letters to uppercase.

The following table gives a listing of default variables. These are environment variables for CGI applications and HTTP variables for IDC applications.

**Peer Web Services Server Variables**

| Variable | Meaning |
|---|---|
| ALL_HTTP | All HTTP headers that were not already parsed into one of the listed variables. the form HTTP_<*header field name*>, for example:<br><br>`HTTP_ACCEPT: */*, q=0.300, audio/x-aiff, audio/basic, image/jpeg, image/gif`<br>`HTTP_USER_AGENT: Microsoft Internet Explorer/0.1 (Win32)`<br>`HTTP_REFERER: http://webserver/samples/dbsamp/dbsamp3.htm`<br>`HTTP_CONTENT_TYPE: application/x-www-form-urlencoded`<br>`HTTP_CONTENT_LENGTH: 10` |
| AUTH_TYPE | The type of authorization in use. If the user name has been authenticated by the server, this will contain Basic. Otherwise, it will not be present. |
| CONTENT_LENGTH | The number of bytes that the script can expect to receive from the client. |
| CONTENT_TYPE | The content type of the information supplied in the body of a POST request. |
| GATEWAY_INTERFACE | The revision of the CGI (Common Gateway Interface) specification with which this server complies |
| HTTP_ACCEPT | Special-case HTTP header. Values of the Accept: fields are concatenated, separated by comma (,); for example, if the following lines are part of the HTTP header:<br><br>`accept: */*; q=0.1`<br>`accept: text/html`<br>`accept: image/jpeg`<br><br>then the HTTP_ACCEPT variable will have a value of:<br><br>`*/*; q=0.1, text/html, image/jpeg` |
| LOGON_USER | The user's Windows NT account. |
| PATH_INFO | Additional path information, as given by the client. This comprises the trailing script name but before the query string (if any). |
| PATH_TRANSLATED | The value of PATH_INFO, but with any virtual path name expanded into a dire |
| QUERY_STRING | The information that follows the question mark (?) in the URL that referenced t |
| REMOTE_ADDR | The IP address of the client. |
| REMOTE_HOST | The hostname of the client. |
| REMOTE_USER | The user name supplied by the client and authenticated by the server. |
| REQUEST_METHOD | The HTTP request method. |
| SCRIPT_NAME | The name of the script program being executed. |

| | |
|---|---|
| SERVER_NAME | The server's hostname (or IP address) as it should appear in self-referencing |
| SERVER_PORT | The TCP/IP port on which the request was received. |
| SERVER_PORT_SECURE | The value of 0 or 1. The value 1 indicates the request is on the encrypted port. |
| SERVER_PROTOCOL | The name and version of the information-retrieval protocol relating to this requ |
| SERVER_SOFTWARE | The name and version of the Web server under which the Internet Server Ext |
| URL | The URL of the request. |

Contents Index




# Complete List of HTML Tags

Microsoft® Internet Explorer 2.0 adds many new features which are great for HTML authors. This document describes HTML tags supported by Internet Explorer version 2.0.

If you have problems viewing the information in the tables, you can modify your Microsoft Internet Explorer 2.0 font size by clicking the Fonts menu, clicking View, and changing your current selection.

Be sure to check out our **Volcano Coffee Company and Creating Great Pages Tour** that highlights detailed examples of tags you can use to create cool pages.

---

## List of HTML Features Supported by Internet Explorer 2.0

---

### A

Stands for *anchor* and specifies a hypertext link.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| HREF="*URL*" | Specifies a destination address. Everything between the <A HREF...> and the </A> (both text and pictures) becomes a clickable hyperlink to that address. | <A HREF="http://www.microsoft.com"> This is a link to Microsoft.</A> | HTML 2 |
| HREF="*filename*" | Specifies a destination file. The server looks in the directory that the current document is stored in to find the file. | <A HREF="home.htm">This is a link to a file called Home.htm in the same directory as this page.</A> | HTML 2 |
| NAME="*name*" | Specifies an anchor point within an HTML document. | Anchor: <A NAME="map1"> Links that point to that anchor: <A HREF="#map1">...</A>, <A HREF="http://www.sample.com/home.htm#map1">...</A> | HTML 2 |

---

### ADDRESS

Renders text as italics. See also Cite, Dfn, Em, I.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Changes text to italic. | <ADDRESS>Hi there!</ADDRESS> | HTML 2 |

## AREA

Specifies the shape of a "hot spot" in a client-side image map.

Client-side image maps are pictures which cause the browser to jump to different URLs depending on where you click. See **Client Side Image Maps** for a full explanation.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| COORDS="*x1, y1, x2, y2, ...*" | Coordinates that define the hot spot's shape. RECT hot spots, for example, use just two. | <AREA SHAPE="RECT" COORDS="50, 25, 150, 125" HREF="http://www.sample.com"> | IExplore |
| HREF="*URL*" | Specifies the destination of the hot spot. | *see above* | IExplore |
| NOHREF | Indicates that clicks in this region should cause no action. | <AREA SHAPE="RECT" COORDS="50, 25, 150, 125" NOHREF> | IExplore |
| SHAPE="*shape type*" | Denotes the type of shape. Allowed values: RECT, RECTANGLE, CIRC, CIRCLE, POLY, or POLYGON. (CIRC/CIRCLE takes three coordinates, *centerx*, *centery*, and *radius*; POLY/POLYGON takes three or more pairs of coordinates denoting a polygonal region.) | Rectangle: <AREA SHAPE="RECT" COORDS="50, 25, 150, 125" HREF="http://www.sample.com"> creates a rectangular hot spot from (50, 25)to (150, 125) | IExplore |

## B

Renders text in boldface. See also Strong.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Changes text to bold. | <B>Hi there!</B> | HTML 2 |

## BASE

Specifies document's URL.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| HREF="*URL*" | Specifies the document's full URL in case the document gets read out of context and the reader wants to refer to the original. | <BASE HREF="http://www.sample.com/hello.htm"> | HTML 2 |

## BASEFONT

Sets base font value.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| SIZE=*n* (*n* is between 1 | Sets the base font size. Throughout the | <BASEFONT SIZE=3> | Netscape |

| | | | |
|---|---|---|---|
| SIZE=*n* (*n* is between 1 and 7 inclusive; default is 3; 7 is largest) | Sets the base font size. Throughout the document, relative font size settings -- for example, <FONT SIZE=+3> -- are set according to this. | <BASEFONT SIZE=3> This sets the base font size to 3. <FONT SIZE=+4> Now the font size is 7. <FONT SIZE=-1> Now the font size is 2. | Netscape |

## BGSOUND

The new BGSOUND tag allows you to create pages with background sounds or "soundtracks." The sound you heard when you opened this page is a background sound. (Choose Refresh from the View menu to hear it again.) Sounds can either be samples (.wav or .au format) or MIDI (.mid format).

For an example, go to **Multimedia and Images**.

Here are the attributes associated with BGSOUND.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| SRC=*URL* | Specifies the address of a sound to be displayed. | <BGSOUND SRC="boing.wav">You will hear a boinging noise as soon as Internet Explorer has downloaded the file Boing.wav. | IExplore |
| LOOP=*n* LOOP=INFINITE | Specifies how many times a sound will loop when activated. If *n*=-1, or if LOOP=INFINITE is specified,it will loop indefinitely. | <BGSOUND SRC="boing.wav"LOOP=5>You will hear a boinging noise five times in a row. <BGSOUND SRC="boing.wav" LOOP=INFINITE>You will hear boinging noises as long as the page is active. | IExplore |

## BLOCKQUOTE

Sets apart a quotation in text.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Indents both left and right margins. Used to set apart quotations in text. | <BLOCKQUOTE>Hi there! This is a lot of text ... and this is the end of it. </BLOCKQUOTE> | HTML 2 |

## BODY

Specifies beginning and end of document body.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Indicates where the body of an HTML document begins and ends. | <HTML> <BODY>Here's a Web page!</BODY></HTML> | HTML 2 |
| BACKGROUND="*URL*" | Specifies a background picture. The picture is tiled behind the text and graphics on the page. For an example of a background picture, go to **Basics**. | <BODY BACKGROUND="linoleum.gif"> ... </BODY> | HTML 3 |

| | | | |
|---|---|---|---|
| BGCOLOR= *#rrggbb* or *colorname* | Sets the background color of the page. *rrggbb* is a hexadecimal number denoting a red-green-blue color value (the number sign is optional). BGCOLOR and each of the other color attributes here can also be set to a colorname. See **Using Color**. | <BODY BGCOLOR=#ff0000>This page has a red background.</BODY> *or* <BODY BGCOLOR=RED>This page also has a red background.</BODY> | Netscape |
| BGPROPERTIES= *fixed* | Specifies a watermark, which is a background picture that does not scroll. | <BODY BACKGROUND="linoleum.gif" BGPROPERTIES=FIXED>The background on this page is fixed so that it does not scroll.</BODY> | IExplore |
| LEFTMARGIN=*n* | Specifies the left margin for the entire body of the page and overrides the default margin. Set to "0", the left margin will be exactly on the left edge. | <BODY LEFTMARGIN="60" TOPMARGIN="0">The left margin of this page is 60 pixels wide.</BODY> | IExlpore |
| LINK=*#rrggbb* or *colorname* | Sets the color of links that have not yet been visited. | <BODY LINK=#0000ff>This page has blue links...</BODY> | Netscape |
| TEXT=*#rrggbb* or *colorname* | Sets the color of text on the page. | <BODY TEXT=FUCHSIA>This text is fuchsia.</BODY> | Netscape |
| TOPMARGIN=*n* | Specifies the margin for the top of the page and overrides the default margin. Set to "0" the top margin will be on the precise top edge. | <BODY LEFTMARGIN="60" TOPMARGIN="0">The top margin of this page is 0 pixels high.</BODY> | IExlpore |
| VLINK=*#rrggbb* or *colorname* | Sets color of links that have already been visited. | <BODY VLINK=#00ff00>This page has green visited links...</BODY> | Netscape |

## ✓ BR

Inserts a line break.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Inserts a line break. | <BR> | HTML 2 |
| CLEAR=LEFT, RIGHT, *or* ALL | Inserts vertical space so that the next text displayed will be past left- or right-aligned "floating" images. LEFT inserts space so that the next text appears aligned with the left margin directly below a left-aligned floating image; RIGHT is the same, but for the right side; and ALL puts the next text past all floating images. | <IMG SRC="sample.gif" ALIGN=LEFT> Here's some text to the right of a picture. <BR CLEAR=LEFT>Here's some text beneath the picture. | HTML 3 |

## ✓ CAPTION

Specifies a caption for a table and must be used within the TABLE tag.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| ALIGN=LEFT, RIGHT, *or* CENTER | Draws the caption left-aligned, right-aligned, or centered with the table borders. | <TABLE><CAPTION ALIGN=CENTER>This caption will be centered between the left and right borders of the table.</CAPTION><TR><TD>This is a cell in the table.</TD></TR></TABLE> | HTML 3 |
| VALIGN=TOP, *or* BOTTOM | Draws the caption on the top or the bottom of the table. | <TABLE><CAPTION ALIGN=CENTER VALIGN=BOTTOM>This caption will appear centered below the table.</CAPTION><TR><TD>This is a cell in the table.</TD></TR></TABLE> | HTML 3 |

## CENTER

Causes subsequent text and images to be centered.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Causes subsequent text and images to be centered. | <CENTER>Hi there!</CENTER> | Netscape |

## CITE

Renders text in italics. See also Address, Dfn, Em, I.

## CODE

Specifies a code sample. See also Samp.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Renders text in a small font. (If no FONT FACE is specified, the font used is fixed-width.) | <CODE>Here is some text in a small fixed-width font.</CODE> | HTML 2 |

## COMMENT

Indicates a comment.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Indicates a comment. The text between the tags is ignored, unless it contains HTML code. | <COMMENT>This won't be printed.</COMMENT> | HTML 2 |

## DD

Specifies a definition in a definition list.

| Attribute | Explanation | Example | Source |
|---|---|---|---|

| | | | |
|---|---|---|---|
| *none* | Indicates that the text is a definition of a term and should therefore be displayed in the right-hand column of a definition list. | See DL. | HTML 2 |

## DFN

Renders text in italics. See also Address, Cite, Em, I.

## DIR

Denotes a directory list.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Specifies that the following block consists of individual items, each beginning with an <LI> tag and none containing more than 20 characters, which should be displayed in columns. | <DIR> <LI>Art<br><LI>History<br><LI>Literature<br><LI>Sports<br><LI>Entertainment<br><LI>Science</DIR> | HTML 2 |

## DL

Denotes a definition list.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Specifies that the following block is a definition list: that is, an automatically formatted two-column list with terms on the left and their definitions on the right. | <DL> <DT>Cat<DD>A furry cute animal that purrs and likes milk.<br><DT>Lizard<DD>A weird desert animal with a long tongue.</DL> | HTML 2 |

## DT

Specifies a term in a definition list.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Indicates that the text is a term to be defined and should therefore be displayed in the left-hand column of a definition list. | See DL. | HTML 2 |

## EM

Renders text in italics. See also Address, Cite, Dfn, I.

## FONT

Changes the font.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| COLOR=#*rrggbb or* COLOR=*color name* | Sets font color. *rrggbb* is a hexadecimal number denoting a | <FONT COLOR=#ff0000>This text is red.</FONT> | IExplore |

| | | | |
|---|---|---|---|
| COLOR=*color name* | hexadecimal number denoting a red-green-blue color value (the number sign is optional). Can also be set to a colorname. See **Using Color**. | is red.</FONT> *or* <FONT COLOR=RED>This text is also red.</FONT> | |
| FACE="*name [,name2] [,name3]*" | Sets the font. A list of font names can be specified. If the first font is available on the system, it will be used, otherwise the second will be tried, and so on. If none are available, a default font will be used. | <FONT FACE="Arial,Lucida Sans,Times Roman"> This text will be in either Arial, Lucida Sans, or Times Roman, depending on which fonts you have installed on your system.</FONT> | IExplore |
| SIZE=*n* SIZE=+*n or* -*n* | Specifies font size between 1 and 7 (7 is largest). A plus or minus before the number indicates a size relative to the current BASEFONT setting. (See BASEFONT.) *Note:* Relative font sizes are not cumulative. Putting two<FONT SIZE=+1> tags in a row does not result in the font size being increased by 2. See example. | <BASEFONT SIZE=3> This sets the base font size to 3.<br><FONT SIZE=+4> Now the font size is 7.<br><FONT SIZE=-1> Now the font size is 2.<br><FONT SIZE=+2>Now the font size is 5.<br><FONT SIZE=1>Now the font size is 1. | Netscape |

## FORM

Denotes a form.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| ACTION="*URL*" | Specifies the address to be used to carry out the action of the form. If none is specified, the base URL of the document is used. | <FORM ACTION="http://www.sample.com/bin/search"> ... </FORM> | HTML 2 |
| METHOD=GET *or* POST | Indicates how the form data should be sent to the server. GET means append the arguments to the action URL and open it as if it were an anchor; POST means send the data via an HTTP post transaction. | <FORM ACTION="http://www.sample.com/bin/search" METHOD=GET> ... </FORM> | HTML 2 |

## H*n*

Renders text in heading style.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Use H1 through H7 to specify different sizes and styles of heading.(H1 is the largest.) | <H1>Welcome to Internet Explorer!</H1> | HTML 2 |
| ALIGN=CENTER | Centers the heading. | <H2 ALIGN=CENTER>How to Use Internet Explorer</H2> | HTML 3 |

## HR

Draws a horizontal rule.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Draws a horizontal rule the width of the window. | <HR> | HTML 2 |
| ALIGN=LEFT, RIGHT, or CENTER | Draws the rule left-aligned, right-aligned, or centered. | <HR ALIGN=CENTER> | HTML 3 |
| COLOR | Adds color to the rule. | <HR COLOR=#rrggbb or *colorname*> | IExplore |
| NOSHADE | Draws the rule without 3-D shading. | <HR NOSHADE> | Netscape |
| SIZE | Sets the height of the rule in pixels. | <HR SIZE=5> | Netscape |
| WIDTH=*n%* | Sets the width of the rule as a percentage of window width. | <HR WIDTH=50%> | Netscape |
| WIDTH=*n* | Sets the width of the rule in pixels. | <HR WIDTH=250> | Netscape |

## HTML

Denotes the file is an HTML document.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Denotes the file is an HTML document. | <HTML><P>This is an HTML document.</P></HTML> | HTML 2 |

## I

Renders text in italics. See also Address, Cite, Dfn, Em.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Renders text in italics. | <I>This text will be in italics.</I> | HTML 2 |

## IMG

Microsoft Internet Explorer 2.0 enables you to embed .avi (Audio Video Interleave) video clips in HTML pages. This is done by adding several new attributes, notably DYNSRC (Dynamic Support), to the IMG tag. Non–video-enabled browsers display still images in their place.

For an example, go to **Multimedia and Images**.

Here is a complete description of the new attributes.

| Attribute | Explanation | Example | Source |
|---|---|---|---|

| | | | |
|---|---|---|---|
| ALIGN=TOP, MIDDLE, *or* BOTTOM | The surrounding text is aligned with the top, middle, or bottom of the picture. | <IMG SRC="sample.gif" ALIGN=MIDDLE>This text is aligned with the middle of the graphic named Sample. | HTML 3 |
| ALIGN=LEFT, *or* RIGHT | The picture is drawn as a left-aligned or right-aligned "floating image," and text will flow around it. (See BR CLEAR.) | <IMG SRC="sample.gif" ALIGN=LEFT>This text will appear to the right of the graphic named Sample. | HTML 3 |
| ALT="*text*" | Specifies text that will be displayed in place of the picture if Show Pictures is turned off. | <IMG SRC="sample.gif" ALT="Picture of the Taj Mahal"> | HTML 2 |
| BORDER=*n* | Specifies the size of a border to be drawn around the image. If the image is a hyperlink, the border is drawn in the appropriate hyperlink color. If the image is not a hyperlink, the border is invisible. | <A HREF="sample.url"><IMG SRC="test.gif" BORDER=5>That was an image with a five-pixel-wide colored border around it.</A> | Netscape |
| CONTROLS | If a video clip is present, a set of controls is displayed under the clip. | <IMG DYNSRC="fun.avi" CONTROLS> The above video has a set of transport controls under it. | IExplore |
| DYNSRC=*URL* | Specifies the address of a video clip or VRML world to be displayed in the window. Stands for Dynamic Support. | <IMG SRC="sample.gif" DYNSRC="test.avi"> If your browser Sources inline video, you will see the movie Test.avi; otherwise you will see the picture Sample.gif. | IExplore |
| HEIGHT=*n* | Along with WIDTH, specifies the size at which the picture is drawn. (If the picture's actual dimensions differ from those specified with WIDTH and HEIGHT, the picture is stretched to match what's specified.) Internet Explorer also uses this to draw a placeholder of appropriate size for the picture before it's loaded. | <IMG SRC="sample.gif" WIDTH=320 HEIGHT=200> | HTML 3 |
| HSPACE=*n* | Along with VSPACE, specifies margins for the image. Similar to BORDER, except the margins are not painted with color when the image is a hyperlink. | <IMG SRC="sample.gif" HSPACE=5 VSPACE=10> This image has five pixels of space on its left and right, and 10 pixels of space above and below. | Netscape |
| ISMAP | Identifies the picture as a server-side image map. Clicking the picture transmits the coordinates of the click back to the server, triggering a jump to another page. | The following inserts the picture Sample.gif into the document and indicates that when the user clicks it, the server should interpret the click according to the information in Jump.map: <A HREF="jump.map"> <IMG SRC="sample.gif" ISMAP></A> | HTML 2 |

| | | | |
|---|---|---|---|
| LOOP=*n*<br>LOOP=INFINITE | Specifies how many times a video clip will loop when activated. If *n*=-1, or if LOOP=INFINITE is specified, it will loop indefinitely. | <IMG SRC="preview1.gif" DYNSRC="movie.avi" LOOP=3><br>The above video will loop three times when activated.<br><br><IMG SRC="preview1.gif" DYNSRC="movie.avi" LOOP=INFINITE><br>The above video will loop indefinitely until stopped. | IExplore |
| SRC | Specifies the address of the picture to insert. | <IMG SRC="http://www.sample.com/test.gif"> | HTML 2 |
| START=<br>FILEOPEN *and/or*<br>MOUSEOVER | For video clips: specifies when the file should start playing. FILEOPEN means start playing as soon as the file is done opening. This is the default. MOUSEOVER means start playing when the user moves the mouse cursor over the animation. | <IMG SRC="preview1.gif" DYNSRC="skiing.avi" START=FILEOPEN><br>The above video will start playing as soon as it is opened.<br><IMG SRC="preview2.gif" DYNSRC="curling.avi" START=MOUSEOVER LOOP=5><br>The above video will start playing when the user moves the mouse over it, and will loop five times before stopping. | IExplore |
| | The user can specify both together. | <IMG SRC="preview3.gif" DYNSRC="windsurfing.avi" START=MOUSEOVER, FILEOPEN><br>The above video will play once as soon as it opens and thereafter will play whenever the user moves the mouse over it. | |
| USEMAP "*map name*" | Identifies the picture as a client-side image map and specifies a MAP to use for acting on the user's clicks. See **Client Side Image Maps** for a code example. | <IMG SRC="sample.gif" USEMAP "#map1"> IExplore | IExplore |
| VSPACE=*n* | See HSPACE. | <IMG SRC="sample.gif" HSPACE=5 VSPACE=10> This image has five pixels of space on its left and right, and 10 pixels of space above and below. | Netscape |
| WIDTH=*n* | See HEIGHT. | <IMG SRC="sample.gif" WIDTH=320 HEIGHT=200> | HTML 3 |

## INPUT

Specifies a form control.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| ALIGN=TOP, MIDDLE, *or* BOTTOM | Used when TYPE=IMAGE. Specifies how the next line of text will be aligned with the image. | <INPUT NAME="Control1" TYPE=IMAGE SRC="sample.gif" ALIGN=MIDDLE>This text is aligned with the middle of the preceding image. | HTML 2 |
| CHECKED=TRUE *or* FALSE | For check boxes and radio buttons, indicates that they are selected. | <INPUT NAME="Control2" TYPE=CHECKBOX CHECKED=TRUE> | HTML 2 |
| MAXLENGTH= "*length*" | Indicates the maximum number of characters that can be entered into a text control. | <INPUT NAME="Control3" TYPE=TEXTBOX MAXLENGTH=10> | HTML 2 |
| NAME="*name*" | Specifies the name of the control. | <INPUT NAME="Control4" TYPE=TEXTBOX MAXLENGTH=10> | HTML 2 |
| SIZE="*size*" SIZE="*width, height*" | Specifies the size of the control (in characters). For TEXTAREA-type controls, both height and width can be specified. | <INPUT NAME="Control5" TYPE=TEXTBOX SIZE=30><INPUT NAME="Control6" TYPE=TEXTAREA SIZE="30,5"> | HTML 2 |
| SRC="*address*" | Used when TYPE=IMAGE. Specifies the address of the image to be used. | <INPUT NAME="Control7" TYPE=IMAGE SRC="sample.gif" ALIGN=MIDDLE> | HTML 2 |
| TYPE | Specifies what type of control to use. See **Form Control Types**. | <INPUT NAME="Control8" TYPE=CHECKBOX CHECKED=TRUE> | HTML 2 |
| VALUE | For textual/numerical controls, specifies the default value of the control. For Boolean controls, specifies the value to be returned when the control is turned on. | <INPUT NAME="Control9" TYPE=TEXTBOX VALUE="Your Name"> <INPUT NAME="Control10" TYPE=CHECKBOX VALUE=TRUE> | HTML 2 |

## ISINDEX

Indicates the presence of a searchable index.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Displays the following message followed by a textbox: "You can search this index. Type the keyword(s) you want to search for:" When the user enters text and presses ENTER, that text is posted back to the page's URL as a query. | <ISINDEX> | HTML 2 |
| ACTION=*filename* | Specifies the gateway program to which the string in the text box should be passed. | <ISINDEX ACTION="search"> | Netscape |
| PROMPT="*prompt text*" | Specifies a prompt to be used instead of the above. | <ISINDEX PROMPT="Type in keywords here"> | HTML 3 |

## KBD

Renders text in fixed-width and boldface type.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Renders text in fixed-width and boldface type. | <KBD>this is text that the user is supposed to type</KBD> | HTML 2 |

## LI

Denotes one item of a list.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | In a <UL>, <OL>, <DIR>, or <MENU> block, denotes a new list item. | <UL> <LI>This is the first item<br><LI>This is the second item<br></UL> | HTML 2 |
| TYPE=A, a, I, i, *or* 1 | Changes the style of an ordered list. Codes: A=use large letters; a=use small letters; I=use large Roman numerals; i=use small Roman numerals; 1=use numbers. | <OL> <LI TYPE=A>This is item A.<br><LI TYPE=1>This is item 2.<br><LI TYPE=i>This is item iii.<br></OL> | Netscape |
| VALUE=*n* | Changes the count of ordered lists as they progress. | <OL> <LI>This is item #1.<br><LI VALUE=3>This is item #3.</OL> | Netscape |

## LISTING

Renders text in fixed-width type. See also Pre, Tt, Xmp.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Renders text in fixed-width type. | <LISTING>Here's some plain text.</LISTING> | HTML 2 |

## MAP

Specifies a collection of hot spots for a client-side image map.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| NAME | Gives the MAP a name so it can be referred to later. See **Client Side Image Maps** for an example of a client-side image map. | <MAP NAME="map1"> <AREA ...> <AREA...> </MAP> | IExplore |

## MARQUEE

The new MARQUEE tag enables you to create a scrolling text marquee. Here is an example of one:

The HTML entry used to insert the marquee above is:

<MARQUEE BGCOLOR=#EFD6C6 DIRECTION=RIGHT BEHAVIOR=SCROLL SCROLLAMOUNT=10 SCROLLDELAY=200><FONT COLOR="WHITE">This is a scrolling marquee.</FONT></MARQUEE>

Marquees can be left- or right-aligned, like images, and have a variety of attributes to control them.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Makes a scrolling text marquee. | <MARQUEE>This text will scroll!</MARQUEE> | IExplore |
| ALIGN=TOP, MIDDLE, *or* BOTTOM | Specifies that the text around the marquee should align with the top, middle, or bottom of the marquee. | <MARQUEE ALIGN=TOP>The following words, "Hi there!", will be aligned with the top of this marquee.</MARQUEE> Hi there! | IExplore |
| BEHAVIOR= SCROLL, SLIDE, *or* ALTERNATE | Specifies how the text should behave. SCROLL (the default) means start completely off one side, scroll all the way across and completely off, and then start again. SLIDE means start completely off one side, scroll in, and stop as soon as the text touches the other margin. ALTERNATE means bounce back and forth within the marquee. | <MARQUEE BEHAVIOR=SCROLL>This text will scroll all the way on and then all the way off.</MARQUEE> <MARQUEE BEHAVIOR=SLIDE>This marquee will scroll in and "stick."</MARQUEE> <MARQUEE BEHAVIOR=ALTERNATE>This text will bounce back and forth.</MARQUEE> | IExplore |
| BGCOLOR= *#rrggbb* or *colorname* | Specifies a background color for the marquee, either as a RGB triple or using a "friendly" colorname. (See **Using Color** for a list of valid colornames.) | <MARQUEE BGCOLOR=#FF0000>This marquee has a red background!</MARQUEE> | IExplore |
| DIRECTION=LEFT *or* RIGHT | Specifies which direction the text should scroll. The default is LEFT, which means scrolling to the left from the right. | <MARQUEE DIRECTION=RIGHT>This marquee will scroll from the left in a rightward direction.</MARQUEE> | IExplore |
| HEIGHT=*n or* HEIGHT=*n%* | Specifies the height of the marquee, either in pixels or as a percentage of the screen height. | <MARQUEE HEIGHT=50% WIDTH=80%>This marquee is half the height of the screen and 80% of the width.</MARQUEE> | IExplore |
| HSPACE=*n* | Specifies left and right margins for the outside of the marquee, in pixels. | <MARQUEE HSPACE=10 VSPACE=10>This marquee will be separated from the surrounding text by a 10-pixel border.</MARQUEE> | IExplore |

| | | | |
|---|---|---|---|
| LOOP=*n*<br>LOOP=INFINITE | Specifies how many times a marquee will loop when activated. If *n*=-1, or if LOOP=INFINITE is specified, it will loop indefinitely. | <MARQUEE LOOP=5>This marquee will loop five times.</MARQUEE> | IExplore |
| SCROLLAMOUNT=*n* | Specifies the number of pixels between each successive draw of the marquee text. | <MARQUEE SCROLLDELAY=5 SCROLLAMOUNT=50>This is a very fast marquee.</MARQUEE> | IExplore |
| SCROLLDELAY=*n* | Specifies the number of milliseconds between each successive draw of the marquee text. | <MARQUEE SCROLLDELAY=5 SCROLLAMOUNT=50>This is a very fast marquee.</MARQUEE> | IExplore |
| VSPACE=*n* | Specifies top and bottom margins for the outside of the marquee, in pixels. | <MARQUEE HSPACE=10 VSPACE=10>This marquee will be separated from the surrounding text by a 10-pixel border.</MARQUEE> | IExplore |
| WIDTH=*n or*<br>WIDTH=*n%* | Sets the width of the marquee, either in pixels or as a percentage of the screen width. | <MARQUEE HEIGHT=50% WIDTH=80%>This marquee is half the height of the screen and 80% of the width.</MARQUEE> | IExplore |

## MENU

Denotes a list of items.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Specifies that the following block consists of individual items, each beginning with an <LI> tag. | <MENU><LI>This is the first item in the menu.<br><LI>And this is the second item in the menu.</MENU> | HTML 2 |

See DIR

## META

Microsoft Internet Explorer Sources client pull using the META tag. The META tag must be inside the HEAD tag of the HTML document. The URL attribute requires a fully qualified URL (for example, http://www.sample.com/reload.htm).

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| HTTP-EQUIV= "REFRESH" | Causes a document to be automatically reloaded on a regular basis, specified in seconds. | <HEAD><META HTTP-EQUIV="REFRESH" CONTENT=2><TITLE>Reload Document</TITLE></HEAD> <BODY> <P>This document will be reloaded every two seconds.</BODY> | Netscape |
| CONTENT="*n;* URL=*URL*" | Tells the browser to reload in *n* seconds. If a URL is specified, the browser will load the URL after the time specified has elapsed. If no URL is specified, it will reload the current document. | <HEAD><META HTTP-EQUIV="REFRESH" CONTENT="5; URL=http://www.sample.com/next.htm"> <TITLE>Load Next Document</TITLE></HEAD> <BODY> <P>After five seconds have elapsed, the document "http://www.sample.com/next.htm" will be loaded.</BODY> | Netscape |

## NOBR

Turns off line breaking.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Renders text without line breaks. | <NOBR>Here's a line of text I don't want to be broken ... here's the end of the line.</NOBR> | Netscape |

## OL

Draws lines of text as an ordered list.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Specifies that the following block consists of individual items, each beginning with an <LI> tag. The items are numbered. | <OL> <LI>This is the first item in the list. <LI>And this is the second item in the list.</OL> | HTML 2 |
| START=*n* | Specifies a starting number for the list. | <OL START=3> <LI>This is item number 3.</OL> | Netscape |
| TYPE=A, a, I, i, *or* 1 | Changes the style of the list. Codes: A=use large letters; a=use small letters; I=use large Roman numerals; i=use small Roman numerals; 1=use numbers. | <OL> <LI TYPE=A>This is item A. <LI TYPE=1>This is item 2. <LI TYPE=i>This is item iii.</OL> | Netscape |

## OPTION

Denotes one choice in a list box.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | In a <SELECT> block, denotes one of the choices that will appear in the list. | See SELECT. | HTML 2 |
| SELECTED | Indicates that this item is the default. If not present, item #1 becomes the default. | See SELECT. | HTML 2 |
| VALUE | Indicates the value that will be returned if this item is chosen. | See SELECT. | HTML 2 |

## P

Denotes a paragraph.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| none | Inserts a paragraph break and denotes a paragraph. The ending tag, </P>, is optional. | <P>This is a paragraph.</P> | HTML 2 |
| ALIGN=CENTER | Centers the paragraph. | <P ALIGN=CENTER>This is a centered paragraph.</P> | HTML 3 |

## PLAINTEXT

Renders text in fixed-width type without processing tags.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Renders text in fixed-width type. Also turns off HTML parsing until the browser encounters the </PLAINTEXT> tag. | <PLAINTEXT> Here's a sample of HTML: <A HREF="sample.url">This is a shortcut to sample.</A></PLAINTEXT> | HTML 2 |

## PRE

Renders text in fixed-width type. See also Listing, Tt, Xmp.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Renders text in fixed-width type. | <PRE>Here's some plain text</PRE> | HTML 2 |

## S

Renders text in strikethrough type.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Renders text in strikethrough type. | <S>This text has a line through it.</S> | HTML 2 |

## SAMP

Specifies a code sample. See also Code.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Renders text in a small font. (If no FONT FACE is specified, the font used is fixed-width.) | <SAMP>Here is some text in a small fixed-width font.</SAMP> | HTML 2 |

## SELECT

Denotes a list box or dropdown list.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Specifies a list box or dropdown list. | <SELECT NAME="Cars" MULTIPLE SIZE="1"><br><OPTION VALUE="1">BMW<br><OPTION VALUE="2">PORSCHE<br><OPTION VALUE="3"<br>SELECTED>MERCEDES</SELECT> | HTML 2 |
| MULTIPLE | Indicates that multiple items can be selected. | See above. | HTML 2 |
| NAME | Specifies a name for the list. | See above. | HTML 2 |
| SIZE | Specifies the height of the list control. | See above. | HTML 2 |

## STRIKE

Renders text in strikethrough type. See S.

## STRONG

Renders text in boldface. See B.

## TABLE

Microsoft® Internet Explorer 2.0 supports HTML tables according to the HTML 3.0 table model. Details of this standard are available at **http://www.w3.org/pub/WWW/TR/WD-tables.html**.

**Note:** This link points to a server that is not under the control of Microsoft Corporation. Please read our **disclaimer** before continuing.

As shown here, Microsoft Internet Explorer 2.0 extends the table specification with color. By adding BGCOLOR=#*nnnnnn* to each opening TD tag, each individual cell can have its own background color. You can also add a caption to the table by using the CAPTION tag.

Example Table

| Hello |
|---|
| there! |

Here is the HTML entry that produces the table shown at right:

```
<TABLE ALIGN=RIGHT BORDER=1 WIDTH=20%>
<CAPTION ALIGN=CENTER VALIGN=BOTTOM><FONT FACE="COURIER NEW" SIZE=1>Example
Table</FONT></CAPTION>
<TR><TD BGCOLOR=#EFD6C6>Hello</TD></TR>
```

<TR><TD BGCOLOR=#FFCC99>there!</TD></TR>
</TABLE>

The following attributes remain with the lowest-level component if they are nested. For example, TH and TD attributes override TR and TABLE attributes, and TR attributes override TABLE attributes. For more information about TABLE, TR, TH, and TD, please see the HTML 3.0 standard mentioned above.

### TABLE, TR, TH, and TD

Here is a complete description of the additional attributes supported by TABLE, TR, TH, and TD.

| Attribute | Explanation | Example | Support |
|---|---|---|---|
| ALIGN=LEFT, *or* RIGHT | Specifies that the table or the text can be left- or right-aligned. The default is left-aligned for TABLE, TR, and TD. The default is center-aligned for TH. | <TABLE ALIGN=RIGHT BORDER=1 width=20%> <TR><TD>This table is right-aligned.</TD></TR></TABLE><br><br><TABLE BORDER=1 width=20%> <TR><TD ALIGN=RIGHT>The text in this cell is right-aligned.</TD></TR></TABLE> | IExplore |
| BACKGROUND=" *URL"* | Specifies a background picture. The picture is tiled behind the text and graphics in the table, table head, or table cell. | <TABLE BACKGROUND="linoleum.gif"> ... </TABLE> | IExplore |
| BGCOLOR= *#rrggbb* or *colorname* | Sets background color. *rrggbb* is a hexadecimal number denoting a red-green-blue color value (the number sign is optional). BGCOLOR and each of the other color attributes here can also be set to a colorname. See **Using Color**. | <TABLE ALIGN=RIGHT BORDER=1 BGCOLOR=RED width=20%> <TR><TD>This table has a red background.</TD></TR></TABLE><br><br><TABLE ALIGN=RIGHT BORDER=1 width=20%> <TR><TD BGCOLOR=RED>This cell has a red background.</TD></TR></TABLE> | IExplore |
| BORDERCOLOR= *#rrggbb* or *colorname* | Sets border color and must be used with the BORDER attribute. *rrggbb* is a hexadecimal number denoting a red-green-blue color value (the number sign is optional). BORDERCOLOR can also be set to a colorname. See **Using Color**. | <TABLE ALIGN=RIGHT BORDER=1 BORDERCOLOR=RED width=20%> <TR><TD>This table has a red border.</TD></TR></TABLE> <TABLE ALIGN=RIGHT BORDER=1 width=20%> <TR><TD BORDERCOLOR=RED>This cell has a red border.</TD></TR></TABLE> | IExplore |

| | | | |
|---|---|---|---|
| BORDERCOLORLIGHT= *#rrggbb* or *colorname* | Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of BORDERCOLORDARK, and must be used with the BORDER attribute. *rrggbb* is a hexadecimal number denoting a red-green-blue color value (the number sign is optional). BORDERCOLORLIGHT can also be set to a colorname. See **Using Color**. | <TABLE ALIGN=RIGHT BORDER=1 BORDERCOLORLIGHT=RED width=20%> <TR><TD>This table has one half of the 3-D border set to red.</TD></TR></TABLE> <TABLE ALIGN=RIGHT BORDER=1 width=20%> <TR><TD BORDERCOLORLIGHT=RED>This cell has one half of the 3-D border set to red.</TD></TR></TABLE> | IExplore |
| BORDERCOLORDARK= *#rrggbb* or *colorname* | Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of BORDERCOLORLIGHT, and must be used with the BORDER attribute. *rrggbb* is a hexadecimal number denoting a red-green-blue color value (the number sign is optional). BORDERCOLORDARK can also be set to a colorname. See **Using Color**. | <TABLE ALIGN=RIGHT BORDER=1 BORDERCOLORDARK=RED width=20%> <TR><TD>This table has one half of the 3-D border set to red.</TD></TR></TABLE> <TABLE ALIGN=RIGHT BORDER=1 width=20%> <TR><TD BORDERCOLORDARK=RED>This cell has one half of the 3-D border set to red.</TD></TR></TABLE> | IExplore |
| VALIGN=TOP, *or* BOTTOM | Specifies that the text can be top- or bottom-aligned. The default is center-aligned. | <TABLE BORDER=1 width=20% VALIGN=TOP><TR><TD>The text in this table is top-aligned.</TD></TR></TABLE> <TABLE BORDER=1 width=20%><TR><TD VALIGN=TOP>The text in this cell is top-aligned.</TD></TR></TABLE> | IExplore |

## TITLE

Specifies a title for the document.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Specifies a title for the document. Internet Explorer uses this for the window caption. | <TITLE>"Welcome To Internet Explorer!"</TITLE> | HTML 2 |

## TT

Renders text in fixed-width type. See also Pre, Listing, Xmp.

## U

Renders text underlined.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Renders text underlined. | <U>This text has a line under it.</U> | HTML 2 |

### UL

Draws lines of text as a bulleted list.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Specifies that the following block consists of individual items, each beginning with an <LI> tag. The items are bulleted. | <UL> <LI>This is the first item in the list. <LI>And this is the second item in thelist.</UL> | HTML 2 |

### VAR

Renders text as a small fixed-width font.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Renders text in a small fixed-width type. | <VAR>Here's some plain text.</VAR> | HTML 2 |

See Samp

### WBR

Inserts a soft linebreak in a block of NOBR text.

| Attribute | Explanation | Example | Source |
|---|---|---|---|
| *none* | Inserts a soft linebreak in a block of NOBR text. | <NOBR> This line of text will not break, no matter how narrow the window gets.<WBR>This one, however<WBR>, will.</NOBR> | Netscape |

### XMP

Renders text in fixed-width type. See also Listing, Pre, Tt.

## Using Color

Microsoft® Internet Explorer 2.0 allows you to use the following colornames. These supported colornames can be used with the BODY, FONT, HR, MARQUEE, and TABLE tags. View the **detailed page on Using Color** for further information.

| SUPPORTED COLORNAMES | | | |
|---|---|---|---|
| Black | White | Green | Maroon |
| Olive | Navy | Purple | Gray |
| Red | Yellow | Blue | Teal |
| Lime | Aqua | Fuchsia | Silver |

**Note:** This feature is not compatible with Netscape v2.0.

---

## HTML Tags Grouped by Category

For further information, see the following topics.

| Category | Description |
|---|---|
| **Basics** | Lists the basic tags you need to create an HTML file. |
| **Client Pull** | Client Pull provides the ability to automatically load a new document in the specified time or reload a document on a regular basis. |
| **Client-Side Image Maps** | The most common use of image maps is to allow users to access different documents by clicking different areas in an image. |
| **Color** | Lists the supported colornames and the tags which support these colornames. |
| **Forms** | Includes tags which produces forms, form control types, and list boxes. |
| **Font Formatting** | Lists the different tags used to format fonts. |
| **Lists** | Includes all of the tags which support different list types. |
| **Multimedia and Images** | Learn how to use tags to embed video clips, sound, and marquees. |
| **Table Support** | Includes the Internet Explorer 2.0 table tag extensions to support HTML tables. |

---

**Back to: HTML Specification Home Page**

# RGB Color Table

New in Internet Explorer 2.0 is the ability to set the background color of individual cells in a table.

I've provided some C code here that will output the HTML code to display a table that lists all possible RGB colors (in increments of 0x33), and sets the "hotlink" reference of each cell to the RGB triplet of the color it represents. This way, all you have to do is move your mouse over the color you want to know the value of, and the RGB triplet will be displayed in the status bar. Cool, huh?

```
void main ()
{
  unsigned int r, g, b,c=0;
  printf ("<HTML><BODY>\n");
  printf ("<H1>RGB Color Table</H1><P>\n");
  printf ("Moving your mouse over any of the colors\n");
  printf ("below will display the RGB color entry\n");
  printf ("for that color in the status bar of your\n");
  printf ("browser.<P>\n");
  printf ("<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0><TR>\n");
  for (r=0; r<=0xFF; r+=0x11) {
    for (g=0; g<=0xFF; g+=0x11) {
      for (b=0; b<=0xFF; b+=0x33) {
        printf ("<TD bgcolor=#%2.2X%2.2X%2.2X>",r,g,b);
        printf ("<A NAME=\"#%2.2X%2.2X%2.2X\" ",r,g,b);
        printf ("HREF=\"#%2.2X%2.2X%2.2X\">",r,g,b);
        printf ("      </A></TD>");
        c++;
        if (c>=24) {
          printf ("</TR><TR>");
          c=0;
        }
        printf ("\n");
      }
    }
  }
  printf ("</TR></TABLE>\n");
  printf ("</BODY></HTML>\n");
}
```

*Robert B. Hess*
*roberth@microsoft.com*

Micros
One Microsof
Redmond WA, 98052