Techniques for Accessible HTML Tables

Stephen Ferg
U.S. Bureau of Labor Statistics
2 Massachusetts Avenue, NE
Washington, DC 20212
Major revision: 2002-08-23 — Minor revision: 2004-04-24

Table of Contents

1 Introduction

1.1 Abstract

This paper discusses techniques for formatting complex tables of statistical data in HTML so that they are accessible to visually impaired users and compliant with the provisions of Section 508. The purpose of the paper is to assist HTML developers, and vendors of screen-reader products, to understand what constitutes "coding to the standard" for accessible HTML tables.

1.2 About the Author

Stephen Ferg is a computer systems analyst in the Division of Data Dissemination Systems (LABSTAT) at the United States Bureau of Labor Statistics (BLS).

1.3 Acknowledgements

This paper was written with support and input from Michael Levi, the LABSTAT Division Chief. I may have written all of the words, but the most basic ideas and the research direction came from Michael. Jo-Ann Yu, a Web specialist in LABSTAT, provided valuable information about how to use CSS stylesheets to control HTML indentation.

1.4 Disclaimer

This paper reflects the author's personal opinions and interpretation of the W3C HTML language specification. The opinions and interpretations expressed in this paper are not in any way endorsed by LABSTAT, by BLS or by the W3C.

2 Some Background on Section 508 and Accessibility

2.1 Introduction

> *On August 7, 1998, President Clinton signed into law the Rehabilitation Act Amendments of 1998 which covers access to federally funded programs and services. The law strengthens section 508 of the Rehabilitation Act and requires access to electronic and information technology provided by the Federal government. The Federal Access Board is responsible for developing accessibility standards for such technology for incorporation into regulations that govern Federal procurement practices.*
>
> Section 508 requires that individuals with disabilities, who are members of the public seeking information or services from a Federal agency, have access to and use of information and data that is comparable to that provided to the public who are not individuals with disabilities, unless an undue burden would be imposed on the agency.
>
> -- text adapted from text on the Access Board web site

Federal agencies that publish data on the Web, do so using HTML, the markup language that underlies all Web pages. HTML Web pages can be made accessible to blind and vision-impaired users by means of "screen reader" technology. A screen reader is a piece of software that reads — that is, speaks — a description of a computer screen. There are a number of screen reader products on the market, including JAWS, Window Eyes, and IBM Home Page Reader.

In order for screen readers to read Web pages effectively, the Web pages must be marked up in HTML using conventions that support interpretation by screen readers. Such conventions require, for example, that pictures embedded in Web pages be accompanied by an ALT attribute that contains text describing the picture. Although a blind user cannot see the picture, a screen reader can at least read the ALT attribute's description of the picture. HTML pages that are marked up in a way that enables them to be read by a screen reader are said to be "accessible" because they are accessible to vision impaired users.

Section 508 requires Federal agencies to make all of their HTML pages accessible to vision-impaired users. This presents a problem for Federal agencies that publish tables of statistical data on the Web. HTML provides only very limited support for tables, so sophisticated table design in HTML is difficult even when the only concern is graphic presentation to sighted users. Developing HTML markup that also supports useful interpretation by screen readers is doubly difficult.

In this situation, in order to comply with Section 508 Federal agencies need specific and detailed descriptions of techniques for creating 508-compliant statistical data tables in HTML. Unfortunately, the guidelines that are currently available are woefully inadequate. This paper is an attempt to address that problem.

This paper discusses techniques for formatting statistical data tables in HTML so that they are accessible to visually impaired users and compliant with the provisions of Section 508. The purpose of the paper is to assist HTML developers, and vendors of screen-reader products, to understand what constitutes "coding to the standard" for accessible HTML tables.

The techniques discussed in this paper are derived — directly or indirectly — from the five basic requirements-statements for 508-compliant HTML.

1. Section 508 itself. The full text of Section 508 is available at the Web site of the Access Board.

   For material related to Web publications, look for paragraph "1194.22 Web-based intranet and Internet information and applications", near the bottom of the document. Subparagraphs (g) and (h) are specifically about HTML tables.

   The Accessibility Board has added a note to this paragraph, which refers to...

2. World Wide Web Consortium Web Content Accessibility Guidelines (usually referred to as "W3C WCAG").

   The guidelines contain checkpoints, and the checkpoints refer to ...

3. An HTML techniques document published by the W3C WCAG as a supplement to the Guidelines.
4. The Access Board has published the official requirements that must be followed by Federal agencies when producing 508-compliant Web pages.
5. The most important reference is the W3C's language specification for HTML 4.01, especially the section on tables. Most of this paper is devoted to close reading and interpretation of this HTML language specification.

Summary of Section 508 requirements for HTML data tables.

Section 508 Provision	WCAG Checkpoint	WCAG Techniques	Access Board Provisions
(g) Row and column	5.1 For example, in HTML, use TD to identify data cells and TH to	5.1.1 Providing	HTML tables

headers shall be identified for data tables.	identify headers.	summary information 5.1.2 Identifying rows and column information	
(h) Markup shall be used to associate data cells and header cells for data tables that have two or more logical levels of row or column headers.	5.2 For data tables that have two or more logical levels of row or column headers, use markup to associate data cells and header cells. [Priority 1] For example, in HTML, use THEAD, TFOOT, and TBODY to group rows, COL and COLGROUP to group columns, and the "axis", "scope", and "headers" attributes, to describe more complex relationships among data.	5.1.1 Providing summary information 5.1.2 Identifying rows and column information	HTML tables

It is important to recognize that accessibility has two different aspects. Only one of these aspects is of concern to Federal agencies in their role of Web content providers.

1. The way that HTML is coded. This aspect is the one that content providers (i.e. Federal agencies) must be concerned with.
2. The way that screen readers read (that is, speak) HTML code. This aspect is the one that vendors of screen reader products must be concerned with.

In order for an end-user to experience a Web page as accessible, both aspects must be addressed: the HTML code must be coded to some standard for accessible code, and the screen-reader must conform to some standard for screen-reader behavior when processing accessible HTML code. Since our concern here is statistical data tables, we propose the following statements for these two standards only as they apply to HTML data tables.

1. We define an *accessible HTML data table* as an HTML data table coded in such a way that all of the table's data cells (i.e. non-empty TD cells) can be mapped to their associated header cells using the algorithms specified in section 11 — and especially section 11.4 (Table rendering by non-visual user agents) — of the HTML language specification.

2. We define a *conforming screen-reader* (with respect to the handling of HTML data tables) as a screen reader that, when processing an accessible HTML data table, is

> capable (1) of mapping each of the data cells in the table to all of its (the cell's) associated header information, using the algorithms specified in the HTML language specification, and (2) of speaking that header information.

As we discuss accessible HTML tables, it is important to remember that HTML tables have two different aspects: the visible format, which is available to sighted users, and the accessible format, which is available to non-sighted users. In this paper we will examine three techniqes for visible formatting, and three for accessible formatting.

3 Some Basic Table Concepts

3.1 Some Definitions

We define a *table* as a presentation format for a set of *data items* and their associated metadata such that:

1. Each data item is stored in a *data cell*
2. The data cells are arranged in a two-dimensional grid called the *table body*
3. The table has a *title* that indicates the nature of the data being presented

We define a *regular table* as a table in which:

1. The metadata information for the data items is presented (1) above the table body in a *table header*, and (2) to the left of the table body in a *table stub*.
2. The metadata in the stub and in the table header is organized into one or more nested levels. Multi-level metadata is is organized into a strict hierarchy (that is, a nested or tree structure; an acyclic directed graph). A metadata hierarchy may be rendered as a tree structure (as table header information usually is) or as an indented outline (as stub information usually is).
3. For any given data cell, all of the cell's metadata are positioned directly above the cell in the table header (as *column headers*) or directly to the left of it in the stub (as *row headers*).
4. The table may contain additional metadata information in the *stub header* cell.
5. The table optionally has a set of *footnotes*. Note that footnotes (like the title) are considered to be part of the table; not information that is external to the table.
6. Any cell of the table may contain zero, one or more *footnote references* (typically, as superscripted numbers) that refer to footnotes.

Table Title				
Stub Header	**Table Header**			
Stub				
Footnotes				

We define an *irregular table* as a table that is not a regular table.

3.2 Examples of Regular Tables

Here are some examples of regular tables.

Example #1: Nested Stubs				
Ruritanian Population Survey		**All Genders**	**By Gender**	
			Males	**Females**
All Regions	**North**	3333	1111	2222
	South	3333	1111	2222

Example #2: Indented Stubs (ROWSPAN)				
Ruritanian Population Survey		**All Genders**	**By Gender**	
			Males	**Females**
All Regions		13332	5555	7777
	North	3333	1111	2222
	South	9999	4444	5555

Example #3: Indented Stubs (CSS)			
Ruritanian Population Survey	**All Genders**	**By Gender**	
		Males	**Females**
All Regions	13332	5555	7777
North	3333	1111	2222
South	9999	4444	5555

3.3 Examples of Irregular Tables

Here are some examples of irregular tables.

1. In this table, the hierarchical structure of the column headers is not top-down. The heading "Gender", which spans two sub-categories of "Male" and "Female", is below "Male" and "Female" rather than above them.

Example #4: Ruritanian Population Survey				
Population		**All Genders**	**Males**	**Females**
			By Gender	
All Regions		13332	5555	7777
By Region	**North**	3333	1111	2222
	South	9999	4444	5555

2.
3. In this table, a header cell occurs below a cell containing data. This irregular table could easily be split into two regular tables.

Example #5: Ruritanian Population Survey		
Population by Region	**Males**	**Females**
North	1111	2222
South	4444	5555
Population by Age	**Below Age 21**	**Age 21 and older**
North	3333	3333
South	9999	3333

In this table, the data cells for North and South span two columns and fall under two column headers.

Example #6: Ruritanian Population Survey		
Ruritanian Population Survey	**Males**	**Females**
All Regions	5555	7777
North	3333	
South	9999	

5. The following table, which is taken from the W3C HTML specification, is irregular because there are data cells whose header cells are not directly above, or directly to the left, of the data cell. For example, the header cell containing "San Jose" is neither immediately to the left of, nor immediately above, the first data cell (containing "37.74").

   This table could be made into a regular indented table by indenting the rows for the dates and subtotals under the rows for the cities.

Travel Expense Report

	Meals	**Hotels**	**Transport**	subtotals
San Jose				
25-Aug-97	37.74	112.00	45.00	
26-Aug-97	27.28	112.00	45.00	
subtotals	65.02	224.00	90.00	379.02
Seattle				
27-Aug-97	96.25	109.00	36.00	
28-Aug-97	35.00	109.00	36.00	
subtotals	131.25	218.00	72.00	421.25
Totals	196.27	442.00	162.00	**800.27**

4 HTML Table Features

Now we turn our attention to an examination of the HTML language, and the features that it offers for creating accessible data tables.

The following discussion is based on the W3C's specification for HTML 4.01, and specifically on Section 11, the section on tables. Within the section on tables, the most important part is 11.4 Table rendering by non-visual user agents. And within that part, the most important subsections are 11.4.1 Associating header information with data cells and 11.4.3 Algorithm to find heading information.

Before we begin, we should note that, in HTML tables, there is a difference between a *position* and a *cell*, and between the *position structure* and the *cell structure* of a table. The distinction is important for understanding how HTML tables work.

4.1 Rows, Columns, and Positions

The *position structure* of an HTML table is a grid of vertical columns and horizontal rows. The intersections of the rows and columns define *positions* in the table. Each position is identified by a pair of coordinates, e.g. as "row 4, column 2".

1,1	1,2	1,3	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	3,4
4,1	4,2	4,3	4,4

Note that in the HTML language specification, references to *row* and *column* refer to rows and columns of positions, not of cells.

4.2 Cells and Cell Types

Visually, a table is divided into *cells.* The way that the table is divided into cells constitutes its *cell structure*.

The position structure of the table — the two-dimensional grid of positions — provides the default cell structure. That is: in the simplest case, each cell covers one position and the cell structure and the position structure of the table are the same.

Cells are defined by TD (table data) and TH (table header) tags.

The original purpose of the distinction between TH and TD tags seems to have been to mark a difference in the type of information in the cells so that the different types of cells could be rendered differently.

> Table cells may contain two types of information: header information and data. This distinction enables user agents to render header and data cells distinctly, even in the absence of style sheets. For example, visual user agents may present header cell text with a bold font. Speech synthesizers may render header information with a distinct voice inflection.

The HTML specification never defines the terms "header information" and "data". Perhaps we can say that what distinguishes header information from data is that a header can "stand by itself" as a piece of information (the name of a category) but that data cannot stand by itself in the same way — it requires some *metadata* contained in one or more header cells. Consider the number 4.5, for instance. It is a piece of data, to be sure. But we have no idea what it *means* until we can associate it with some metadata. Is it the unemployment rate for Chicago in June, 1999? The amount of raw titanium ore refined in Uganda in 1988, measured in units of one hundred thousand tons? Russia's average annual gross national product between 1999 and 2001, measured in units of a quadrillion rubles? A percent change in the Consumer Price Index for last month? Without any metadata, we can't tell.

The essential nature of a data cell, then, is that it cannot stand alone. It must be associated with metadata in one or more header cells to be meaningful. It follows that an essential adjunct of the distinction between data cells and header cells is some notion of a link (connection, association) between a data cell and the header cells that make it meaningful. It is not enough for HTML to provide a way to distinguish data cells from header cells — it must also provide a way to associate a data cell with its header cells.

In a visual browser, this is generally not a problem. When the header cells are enclosed in TH tags and rendered in bold, a sighted user can easily make the connection. He simply looks upward from the data cell to find the bolded cells above it, and he looks left from the data cell to find the bolded cells to its left. Those are the header cells associated with the data cell.

A non-sighted user, on the other hand cannot do this. So in order to support non-sighted users using screen readers, HTML must provide some method that a screen reader can use. We will discuss those features in the section on associating data cells with header cells. But first, let's finish our discussion of data and header cells by making a couple of points.

1. The W3C intended TH cells to be used to contain header information, and TD cells to be used to contain data. For cells that need to act as both data and header cells, TD should be used. along with either the HEADERS or SCOPE algorithm (which we will describe shortly), or AXIS. The HTML language specification gives this example of a table where the row headers are marked up using the TD tag.

Community Courses -- Bath Autumn 1997				
Course Name	**Course Tutor**	**Summary**	**Code**	**Fee**
After the Civil War	Dr. John Wroughton	The course will examine the turbulent years in England after 1646. *6 weekly meetings starting Monday 13th October.*	H27	£32
An Introduction to Anglo-Saxon England	Mark Cottle	One day course introducing the early medieval period reconstruction the Anglo-Saxons and their society. *Saturday 18th October.*	H28	£18
The Glory that was Greece	Valerie Lorenz	Birthplace of democracy, philosophy, heartland of theater, home of argument. The Romans may have done it but the Greeks did it first. *Saturday day school 25th October 1997*	H30	£18

Note the use of the scope attribute with the "row" value. [For example, in **<TD scope="row">The Glory that was Greece</TD>**] Although the first cell in each row contains data, not header information, the scope attribute makes the data cell behave like a row header cell. This allows speech synthesizers to provide the relevant course name upon request or to state it immediately before each cell's content.

2. Note that TH cells were intended to convey header information regardless of whether that header information applies to columns or rows of data. So there are both *column* header cells and *row* header cells. In the following table, all of the shaded cells (all of the cells in the top row, and the leftmost column) are natural candidates to be marked as TH cells.

1,1	**1,2**	**1,3**	**1,4**
2,1	2,2	2,3	2,4
3,1	3,2	3,3	3,4
4,1	4,2	4,3	4,4

4.3 Cell Coverage

The *position structure* of the table — the two-dimensional grid of positions — provides the default *cell structure*. In the simplest case, each cell covers one position.

It is possible, however, to impose a different cell structure on the position structure through the use of the COLSPAN and ROWSPAN attributes of the TD and TH elements. These attributes cause a cell to cover multiple positions. In the following example, Cell A has attributes of COLSPAN = "2" ROWSPAN = "2" that cause it to cover four positions — (1,1), (1,2), (2,1) and (2,2). Cell B covers positions (1,3) and (1,4). And Cell C covers positions (3,1) and (4,1).

<table>
<tr><td colspan="2" rowspan="2">Cell A
COLSPAN="2"
ROWSPAN="2"</td><td colspan="2">Cell B
COLSPAN="2"</td></tr>
<tr><td>2,3</td><td>2,4</td></tr>
<tr><td rowspan="2">Cell C
ROWSPAN="2"</td><td>3,2</td><td>3,3</td><td>3,4</td></tr>
<tr><td>4,2</td><td>4,3</td><td>4,4</td></tr>
</table>

Note that even though a cell may cover multiple positions, each cell is anchored to one particular position — the uppermost, leftmost position that it covers. Cell A is anchored to position (1,1). Cell B is anchored to position (1,3). And cell C is anchored to position (3,1).

4.4 AXIS

A number of discussions of accessibility recommend using AXIS. The W3C's Web Content Accessibility Guidelines Checkpoint 5.2, for example, specifies that

> For data tables that have two or more logical levels of row or column headers, use markup to associate data cells and header cells. [Priority 1] For example, in HTML, use THEAD, TFOOT, and TBODY to group rows, COL and COLGROUP to group columns, and the "axis", "scope", and "headers" attributes, to describe more complex relationships among data.

and Section 5.1.2 of the associated HTML Techniques document says:

> Label table elements with the "scope", "headers", and "axis" attributes so that future browsers and assistive technologies will be able to select data from a table by filtering on categories.

Note, however, that the Access Board's HTML recommendations contain no mention of axis.

4.4.1 AXIS is about possible future technologies

The first, and perhaps most important, thing to note about AXIS is that is *absolutely irrelevant to any browsers that exist today (2002 CE).*

The AXIS attribute was devised when the W3C was in a futuristic mood. Section 5.1.2 looks forward to some future time when browsers will incorporate query-language capabilities that can be used to retrieve data from an HTML table, somewhat in the way that the SQL query language can be used now to retrieve data from a table in a relational database.

That is why the WCAG's HTML techniques document suggests labeling elements with AXIS in order to support "*future* browsers and assistive technologies" [italics mine]. The HTML recommendation is quite clear that browsers with such query-language capabilities do not yet exist, and makes it quite clear that it is making no recommendations about how such query language capabilities might work if they were ever to be devised.

> This specification does not ... make any recommendations about ... how users may query the user agent about this information.

4.4.2 AXIS is about filtering

The second thing to note about AXIS is that the purpose of AXIS is to support a *filtering* capability in some hypothetical future browser query language.

Filtering is a common function of query languages. Suppose you have a relational database with a table named TRAVEL_EXPENSES. This table contains columns LOCATION, DATE, EXPENSE_TYPE, and AMOUNT. If you wanted to see a list of all expenses, you might use the SQL query language to formulate this query:

```
SELECT LOCATION, DATE, EXPENSE_TYPE, AMOUNT FROM TRAVEL_EXPENSES
```

But if you wanted to see just the expenses incurred in Seattle, you'd formulate your query this way:

```
SELECT LOCATION, DATE, EXPENSE_TYPE, AMOUNT FROM TRAVEL_EXPENSES
     WHERE LOCATION = "Seattle"
```

As you can see, the SQL-language mechanism for supporting filtering is the WHERE clause.

Now imagine that we have some future browser that supports a query language with similar filtering capabilities. Suppose that we have an HTML table containing travel expenses. In that table, there is one data cell that contains the value $15, and it is associated with two header cells. One is a column header and one is a row header. The two header cells are defined this way (not in the same row, of course):

```
<TH> San Jose </TH> <TH> Meals </TH>
```

Another data cell contains the value $20, and it is associated with header cells that are defined this way:

```
<TH> Seattle </TH> <TH> Meals </TH>
```

And suppose that we wish to do the same thing that we did with our SQL query — obtain a list of all of the expenses *that were incurred when the location was Seattle.*

That was easy to do in the relational table, because the value "Seattle" occurred in a column named LOCATION. But we can't do the same kind of filtering in the HTML table, because the header cells aren't similarly named. There is, for instance, no way to specify that "Seattle" is a LOCATION rather than an EXPENSE_TYPE like "Meals".

That is, there was no way... until AXIS became part of the HTML language.

The AXIS attribute is defined in section 11.4.2 of the HTML language specification. The section title is "Categorizing Cells", but probably a better title would have been "Categorizing Header Cells", because that is what AXIS does. AXIS provides a way of attaching a *category name* to a header cell. The attribute used to attach a category name to a header cell is *axis.*

Once we have AXIS, we can mark up our header cells with category names, this way:

```
<TH AXIS="LOCATION"> San Jose </TH>
<TH AXIS="LOCATION"> Seattle </TH>

<TH AXIS="EXPENSE_TYPE"> Meals </TH>
```

Once that is done, we have all the information in the table that would be required to answer our query... assuming, of course, that we had a query language in which to express our query.

4.4.3 AXIS is *not* about associating data and header cells

The third thing to note about AXIS is that it is *not* used to create associations between data cells and header cells.

Remember that AXIS is basically a filtering technology. Its purpose is to process a list of associations and filter out the unwanted ones. It works on a list, and the list must already exist in order for it to have something to work on.

Section 11.4.2 of the HTML language specification does not specify how the list of associations is to be created. The discussion of the "Travel Expense Report" example table in section 11.4.2 notes that the associations in the table must have been created via use of ID, HEADERS, and SCOPE attributes. But that is simply because the "Travel Expense Report" table is irregular, and the only practical way to mark up irregular tables is with ID and HEADERS attributes.

So we should note that the use of AXIS is *not* tied to the use of ID and HEADERS attributes. There is nothing in section 11.4.2 that precludes the possibility — at least for regular tables — that the list of associations is created by the algorithm described in section 11.4.1 operating on TD and TH tags.

4.4.4 What does AXIS have to do with accessibility?

Good question. Accessibility and support for a query-language seem to be two quite distinct issues. One can easily imagine future HTML query languages that are entirely visual — the way that SQL is today — and have nothing to do with speech-based rendering. In such a situation, speech rendering would have to be layered on top of the query language in the way that screen readers are layered on top of other computer functions today.

Nevertheless, the two seemed to have been linked in the minds of the authors of section 11.4.2. The section begins: "Users browsing a table with a speech-based agent..." and continues on from there. It looks as if the authors believed that the primary purpose of such a query language would be to assist vision-impaired users in extracting meaningful information from large tables.

4.4.5 Is there any use for AXIS *today*?

The short answer is: Not much. Since no browsers today provide the kind of querying facilities that AXIS is designed to support, there is no real reason for using AXIS today. It is possible that some screen-reader products speak the AXIS attribute if it is present, and that may be useful. But it is not what AXIS was intended for, and there is nothing in the HTML specification that requires a screen-reader to behave this way.

The longer answer is: It depends. The HTML Techniques document puts it quite clearly. It recommends that we...

> Label table elements with the "scope", "headers", and "axis" attributes so that future browsers and assistive technologies will be able to select data from a table by filtering on categories.

If you believe that there is a real possibility that browser technology will be available in the foreseeable future that will offer such querying and filtering capabilities, and you wish to plan now to support it, then you should indeed start labeling your header cells with AXIS.

4.4.6 How To Use AXIS To Make a Data Cell Act Like a Header Cell

There is one possible use for AXIS today. That is to force a TD cell to be treated like a header cell when a screen reader goes searching for the header cells associated with a data cell. This probably falls into the category of "tricks", but it is a trick that the W3C has handed to us on a platter. The last bullet in the description of the BASIC algorithm in section 11.4.3 of the W3C specification for HTML 4.01 says

```
TD cells that set the axis attribute are also treated as header cells.
```

So there are three ways to enable TD cell to supply header information for other cells.

1. Give the cell an ID attribute which can be used in the HEADERS attribute of data cells.
2. Attach a SCOPE attribute to the cell.
3. Attach an AXIS attribute to the cell.

4.5 A final note

We've looked a a number of HTML features that can be useful in formatting tables. We now turn to the topics of *visible formatting* (making a table that looks right to a sighted user) and *accessible formatting* (making a table that a screen reader can effectively read to a non-sighted user).

5 Visible Formatting

We now turn to the topic of formatting data tables for sighted users. The problem here is how to use HTML to make a table that *looks* the way a statistical data table should look.

In most tables of statistical data, there is a fundamental asymmetry in the formatting of the stub and the column headers. If both column and row headers were presented in the same way, as tree-like structures, tables would look like Exhibit A.

The problem with formatting row headers this way is that if the row headers are deeply nested, the table becomes unreasonably wide. For this reason, row headers are not typically presented this way. Instead, they are usually formatted in an indented, outline-like structure (Exhibit B). As in the "All Regions" category in Exhibit B, data cells are often associated with middle and upper nodes of the category structure, as well as with leaf nodes.

Exhibit A

Ruritanian Population Survey			By Gender	
			Males	Females
All Regions			7777	12221
By Region	**North**	**East**	1111	2222
		West	1111	2222
	South	**East**	4444	5555
		West	1111	2222

Exhibit B

Ruritanian Population Survey	By Gender	
	Males	Females
All Regions	5555	7777
North	1111	2222
East	4444	5555
West	4444	5555
South	4444	5555
East	4444	5555
West	4444	5555

It is the use of an indented format in table stubs that causes most of the problems when attempting to visually format data tables in HTML. The current HTML standard (4.01) offers satisfactory facilities for dealing with tree-like formatting, but it has no convenient means of producing indented, outline-style stubs.

The author is currently participating on a Fedstats working group whose goal is to develop a proposal for extensions to the HTML language that will address this problem. Such extensions, however much they may be a part of our future, are not a part of our present. Federal agencies

need to produce accessible HTML data tables *today,* using the facilities available in the current HTML language. For that reason, much of our attention in this paper will be focussed on the problem of how to code tables with indented stubs using the features available in the current HTML language definition.

5.1 Nested Headers

Example #1: Nested Stubs				
Ruritanian Population Survey		**All Genders**	**By Gender**	
			Males	**Females**
All Regions	**North**	3333	1111	2222
	South	3333	1111	2222

Here is the HTML.

```
<TABLE CELLPADDING="5" CELLSPACING="0"
BORDER="1" align="center">
    <CAPTION> Example #1: Nested
Stubs</CAPTION>

    <TR>
    <TH class="center" COLSPAN="2"
ROWSPAN="2"> Ruritanian<BR>
    Population<BR> Survey </TH>

    <TH class="center" ROWSPAN="2">
All <BR> Genders </TH>
    <TH class="center" COLSPAN="2"> By
Gender </TH>
    </TR>

    <TR>
    <TH class="center"> Males </TH>
    <TH class="center"> Females </TH>
    </TR>

    <TR>
    <TH ALIGN="left" ROWSPAN="2"> All
Regions </TH>
    <TH> North </TH>
    <TD ALIGN="right"> 3333 </TD>

    <TD ALIGN="right"> 1111 </TD>
    <TD ALIGN="right"> 2222 </TD>
    </TR>

    <TR>

    <TH> South </TH>
    <TD ALIGN="right"> 3333 </TD>
    <TD ALIGN="right"> 1111 </TD>

    <TD ALIGN="right"> 2222 </TD>
```

```
    </TR>
</TABLE>
```

5.2 Indented Headers using " "

We now turn our attention to techniques for coding tables with indented stubs. Before we go any further, let's take a quick look at a technique that does *not* work, and see why it doesn't work.

It is possible to achieve an indentation effect by prefixing stub heading with one or more iterations of the ` ` (non-breaking space) element. The problem with this technique (which you may be able to see in the table below, depending on your browser settings) is that if the text is too long for the width of the column, it will wrap around to a new line. But when it wraps around, the indentation doesn't carry over to subsequent lines in the cell.

test table

Ruritanian Population Survey	All Genders	By Gender	
		Males	Females
All Regions	13332	5555	7777
North Regions	3333	1111	2222
South Regions	9999	4444	5555

Here is the HTML.

```
<TABLE WIDTH="30%" BORDER="1"
CELLPADDING="5" CELLSPACING="0"
align="center">
    <CAPTION>test table</CAPTION>

    <TR>

    <TH WIDTH="20%" ROWSPAN="2">
Ruritanian <BR> Population <BR>
Survey</TH>

    <TH ROWSPAN="2"> All <BR> Genders
</TH>

    <TH COLSPAN="2"> By Gender </TH>
    </TR>

    <TR>
    <TH> Males </TH>

    <TH> Females </TH>
```

```
    </TR>

    <TR>
    <TH ALIGN="left"> All Regions
</TH>

    <TD ALIGN="right"> 13332 </TD>
    <TD ALIGN="right"> 5555 </TD>
    <TD ALIGN="right"> 7777 </TD>

    </TR>

    <TR>
    <TH ALIGN="left">    
North Regions </TH>
    <TD ALIGN="right"> 3333 </TD>

    <TD ALIGN="right"> 1111 </TD>
    <TD ALIGN="right"> 2222 </TD>
    </TR>

    <TR>

    <TH ALIGN="left">    
South Regions </TH>
    <TD ALIGN="right"> 9999 </TD>
    <TD ALIGN="right"> 4444 </TD>

    <TD ALIGN="right"> 5555 </TD>
    </TR>

</TABLE>
```

5.3 Indented Headers using Cell Control

Example #2: Indented Stubs (ROWSPAN)				
Ruritanian Population Survey		**AllGenders**	**By Gender**	
			Males	**Females**
All Regions		13332	5555	7777
	North	3333	1111	2222
	South	9999	4444	5555

Here is the HTML.

```
<TABLE BORDER="1" CELLPADDING="5"
CELLSPACING="0" ALIGN="CENTER">
  <CAPTION> Example #2: Indented Stubs
(ROWSPAN)</CAPTION>

  <TR>
    <TH COLSPAN="10" ROWSPAN="2">
Ruritanian Population Survey </TH>
    <TH ROWSPAN="2" CLASS="center">
AllGenders </TH>
    <TH COLSPAN="2" CLASS="center"> By
Gender </TH>

  </TR>
  <TR>
    <TH> Males </TH>
    <TH> Females </TH>

  </TR>
  <TR>
    <TH ALIGN="left" COLSPAN="10"> All
Regions </TH>
    <TD ALIGN="right"> 13332 </TD>

    <TD ALIGN="right"> 5555 </TD>
    <TD ALIGN="right"> 7777 </TD>
  </TR>
  <TR>

    <TH ALIGN="left" ROWSPAN="2">
  </TH>
    <TH colspan="9" ALIGN="left">
North </TH>
    <TD ALIGN="right"> 3333 </TD>
```

```
    <TD ALIGN="right"> 1111 </TD>
    <TD ALIGN="right"> 2222 </TD>
  </TR>
  <TR>

    <TH colspan="9" ALIGN="left">
South </TH>
    <TD ALIGN="right"> 9999 </TD>
    <TD ALIGN="right"> 4444 </TD>

    <TD ALIGN="right"> 5555 </TD>
  </TR>
</TABLE>
```

In this technique, the labels running down the left side of the table (the "stub") are contained in a section that is 10 columns wide. For row headers that span the entire stub, we code a single TH cell with COLSPAN="10". For row headers that are indented, we code two TH cells. The first TH cell is empty, and uses ROWSPAN to span the desired number of rows. The second TH cell spans the remaining columns in the stub. So a row specification for a row in which the empty indentation cell must span two rows would be coded like this.

```
<TR> <TH ROWSPAN="2">   </TH>
<TH COLSPAN="9"> North </TH> ...
```

There are a number of variations on this technique, which produce slightly different results. It is possible to get the desired effect using ROWSPAN, or using COLSPAN, or using neither, in the empty cells used to create the indentation. And the table is rendered differently depending on whether the indentation cells contain * * or are completely empty.

5.3.1 Indented tables that use ROWSPAN

Indented Table R1			
		COLUMN A	**COLUMN B**
A		99.9	99.9
	AA	99.9	99.9
		AAA	99.9	99.9
			AAA1	99.9	99.9
			AAA2	99.9	99.9

5.3.2 Indented tables that use COLSPAN

Indented Table C1			
		COLUMN A	**COLUMN B**
A		99.9	99.9
	AA	99.9	99.9
	AAA	99.9	99.9
	AAA1	99.9	99.9
	AAA2	99.9	99.9

5.3.3 Indented tables that don't use ROWSPAN or COLSPAN

This is probably the easiest way to mechanically mark up tables.

Indented Table N1					
				COLUMN A	**COLUMN B**
A				99.9	99.9
	AA			99.9	99.9
		AAA		99.9	99.9
			AAA1	99.9	99.9
			AAA2	99.9	99.9

5.3.4 Indented tables that don't use " "

In all of these tables, you have the option of not using " " in the empty cells. If you leave the cells completely empty, the browser probably will not draw borders around the empty cells. The effect will be a table that looks like this.

Indented Table N2					
				COLUMN A	**COLUMN B**
A				99.9	99.9
	AA			99.9	99.9
		AAA		99.9	99.9
			AAA1	99.9	99.9
			AAA2	99.9	99.9

5.3.5 Problems with this technique

There are two difficulties with techniques that use cell control.

The first is that creating the HTML is quite difficult. Generating such tables programmatically is possible, but creating such tables by hand is far too difficult for a human being.

The second is that the indentation created by the browser may not be uniform. As you can probably see, in many cases the most deeply indented stub is indented by an increment that is noticeably greater than that used for other levels. In theory this could be controlled by careful

specification of WIDTH attributes on each cell, but that brings us back to the first problem — the difficulty of coding creating such HTML.

5.4 Indented Headers using CSS Margin Control

Another technique for producing indented stubs is to have all of the stub text reside in a single left-hand column, and to indent the stub lines using the `margin-left` property of Cascading Style Sheets (CSS). Styles are defined inside a STYLE element in the HEADER element of the HTML file. Here is the code for the style-sheet that defines three properties — sub1, sub2, and sub3 — for different level of indentation.

```
<style type = "text/css">
   .sub1 {margin-left: 1em;}
   .sub2 {margin-left: 2em;}
   .sub3 {margin-left: 3em;}
   td {font-family: monospace; text-
align: right}
   th {text-align: left}
</style>
```

There are a variety of units that you can use to define the amount of indentation that you want, including pixels, millimeters, and "ems". An "em" is the width of a capital "M" in whatever font is currently in use. Since the size of an em will adjust to be appropriate for the context, we've chosen to specifiy our indentation in units of ems.

As you can see, we've also provided styles for TD and TH cells that will govern the horizontal alignment — header text will be left justified and data will be right justified.

The way to use a style is to make it the value of a CLASS attribute. You can do this with almost any HTML element, but when there is no appropriate tag handy, you can use the DIV element. DIV is a multi-purpose tag. In itself, it doesn't do anything — it simply delimits a section of text. Its only purpose, really, is to be a non-entity upon which you can hang stylesheet specification. Here is an example in which the DIV tag refers to one of the classes — sub1 -- that we have just defined in the stylesheet.

```
<div class="sub1"> Some text... </div>
```

Note that style names are case sensitive, so we have invoked the "sub1" class this way — CLASS="sub1" — and not as CLASS="SUB1".

Note also that once a cell is governed by a style, the style can't be over-ridden by HTML specificatons such as ALIGN="CENTER". A style can only be over-ridden by another style, invoked by a CLASS attribute. Left alignment is fine for our *stub* headers, but we want our *column* headers to be centered. To do that, we must create and use a class for center alignment. And just so we have a full repertoire of alignment options at our disposal, we also create a "left" and a "right" style.

```
.center {text-align: center}
.right  {text-align: right}
.left   {text-align: left}
```

Here is an example of a table that uses the CSS styles to control stub indentation. I've made the table unnaturally wide (by specifying WIDTH="60%") so the center alignment of the column header cells can be seen.

Example 2: Metal Prices			
	1998	**1999**	**2000**
Copper	123	123	123
Unrefined	123	123	123
Refined	123	123	123
Iron	123	123	123
Unrefined	123	123	123
Refined	123	123	123
Tin	123	123	123
Unrefined	123	123	123
Refined	123	123	123

Here is the HTML code.

```
<table cellspacing="0" cellpadding="5" border="1" align="center"
width="60%">

<caption> Example 2: Metal Prices </caption>

<tr>
<th> </th>
<th class="center"> 1998 </th>

<th class="center"> 1999 </th>
<th class="center"> 2000 </th>
</tr>

<tr>

<th > Copper </th>
<td > 123 </td>
<td > 123 </td>
<td > 123 </td>

</tr>

<tr>
```

```
<th > <div class="sub1"> Unrefined </div></th>
<td > 123 </td>

<td > 123 </td>
<td > 123 </td>
</tr>

<tr>
<th <div class="sub1"> Refined </div> > </th>

<td > 123 </td>
<td > 123 </td>
<td > 123 </td>
</tr>

... and so on for Iron and Tin ...

</table>
```

6 Accessible Formatting

Now that we have examined techniques for formatting data tables for sighted users, we turn to the topic of *accessible formatting*, that is, formatting tables for non-sighted or vision-impaired users. The problem here is how to use HTML to make a table that a screen reader can effectively read to a vision-impaired user.

More precisely — as we noted earlier — the problem is how to code HTML that conforms to standards for accessible HTML, so that a screen reader that conformed to those standards can effectively read it.

The basic concept underlying accessible HTML tables is that of a list of header cells associated with a data cell.

The way a screen reader makes a data cell accessible is by (1) constructing a list of the header cells associated with the data cell, and then (2) reading the list along with the contents of the data cell. The list is built by the screen reader on the basis of *associations* between the data cell and header cells. The main problem of creating accessible HTML tables, then, is the problem of writing the HTML in such a way that these associations exist, so that a screen reader can build the list from them.

The most complete presentation of the notion of the list, and the process of finding the header cells associated with a data cell, is in section 11.4.3 of the HTML specification.

6.1 The HEADERS Algorithm — using the *HEADERS* and *ID* Attributes

Virtually every HTML element, including the TD and TH tags, can have an ID attribute. The ID attribute is meant to provide a way to uniquely identify some HTML element on a page. Values of the ID attribute are case-sensitive, must begin with a letter (i.e. an alphabetic character), and may not contain spaces. As the HTML specification says:

```
ID ... tokens must begin with a letter ([A-Za-z]) and may be
followed by any number of letters, digits ([0-9]), hyphens ("-"),
underscores ("_"), colons (":"), and periods (".").
```

In this example, we assign the ID "R1" to a header cell.

```
<TH ID="R1"> ... </TH>
```

TD and TH tags can be given a HEADERS attribute. A HEADERS attribute may contain a blank-separated list of IDs. The HEADERS attribute associates the data cell with the cells identified by those IDs.

```
<TD HEADERS="R38 R36 R34 H2 H5"> ... </TH>
```

The HEADERS and ID attributes give the HTML author the maximu possible control over the associations between data cells and header cells, but their use has some drawbacks and limitations.

1. For large tables, the use of HEADERS can significantly increase the size of the HTML code of the table. This can slow download times. How much of a problem this is, depends on the size of the table.
2. The process of adding HEADERS attributes to the data cells can be labor-intensive and error-prone. For large and/or complex tables, adding HEADERS attributes by hand is not feasible — the process must be automated in some way, or it can't reasonably be attempted at all.
3. Some assistive browsers — browsers that read Web pages to vision-impaired users — have a very nice feature for reading HTML tables. As the user moves the cursor left or right from a cell in the table, the browser reads the column headers that have changed with the move but does not read the row headers (which have not changed). And similarly for moving up and down from row to row. But if such a browser reads a table that is marked up only with IDs and HEADERS, it may have no way to provide this kind of service, because the IDs listed in a HEADERS attribute are not divided into column headers and row headers; the HEADERS attribute contains just an undifferentiated list of IDs. A smart enough browser *might* still be able to provide such a service, but certainly

there is nothing in the HTML specification that requires it to provide such a service or specifies how it might be done.

However, despite these limitations, there are two reasons why you might choose to use HEADERS and IDs

First, they are your only option for irregular tables.

The conclusion that you should draw from this, though, is not that you need to use HEADERS and IDs in your irregular tables — it is that you need to restructure your tables so that they are no longer irregular.

In most cases, if an irregular table really does make sense, then it isn't very difficult to restructure it into a regular tables. Some irregular tables are really the concatenation of two logically separate tables. In such cases, the irregular table can easily be split apart into separate, regular, tables. InIn some cases, minor reformatting can turn an irregular table into a regular one. Consider, for example, the widely cited example of the Travel Expense Report in the HTML language specification.

Travel Expense Report

	Meals	Hotels	Transport	subtotals
San Jose				
25-Aug-97	37.74	112.00	45.00	
26-Aug-97	27.28	112.00	45.00	
subtotals	65.02	224.00	90.00	379.02
Seattle				
27-Aug-97	96.25	109.00	36.00	
28-Aug-97	35.00	109.00	36.00	
subtotals	131.25	218.00	72.00	421.25
Totals	196.27	442.00	162.00	**800.27**

It can easily be reformatted as an indented table.

Travel Expense Report

	Meals	Hotels	Transport	subtotals
San Jose				
25-Aug-97	37.74	112.00	45.00	
26-Aug-97	27.28	112.00	45.00	
subtotals	65.02	224.00	90.00	379.02
Seattle				
27-Aug-97	96.25	109.00	36.00	

	28-Aug-97	35.00	109.00	36.00	
	subtotals	131.25	218.00	72.00	421.25
Totals		196.27	442.00	162.00	800.27

The second reason for using HEADERS and IDs is that the current generation of screen readers seems to be able to handle HEADERS and IDs correctly. If your concern is not only to "code to the standard" when writing HTML, but also to work with screen readers that are not yet able to "read to the standard", then using HEADERS and IDs may be your only choice. But it will be a lot of work, and when the day comes that screen readers get smarter, and they can effectively read much simpler markup, you may regret the effort that you expended to mark up your tables with HEADERS and IDs.

6.2 The SCOPE Algorithm

The HTML language specification describes the SCOPE attribute as a sort of labor-saving device. In situations where it is possible to use it — in regularly formatted tables -- SCOPE is functionally equivalent to the HEADERS algorithm, but shorter and easier to use.

> Authors may choose to use [the SCOPE] attribute instead of headers according to which is more convenient; the two attributes fulfill the same function. The headers attribute is generally needed when headers are placed in irregular positions with respect to the data they apply to.

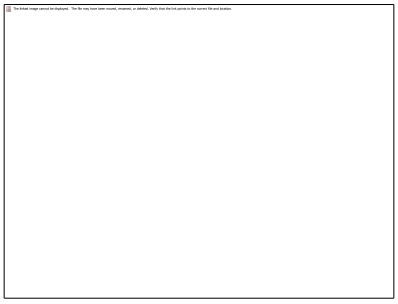
The SCOPE attribute relies on HTML's ability to define groups of columns and rows — the only valid values for the SCOPE attribute are *row*, *col*, *rowgroup*, and *colgroup*. So we must study colgroups and rowgroups in order to understand how SCOPE works.

Reading the HTML specification for colgroups and rowgroups, it is helpful to remember that when the HTML specification was first developed, high-speed Internet connections were not as widely available as they are today. Connections could be slow, and rendering a large data table could be painfully slow — as it still is today, via a modem connection. If a browser always had to have received all of the data in a table before it could start rendering the table on the screen, then the World Wide Web would truly deserve its nickname as the World Wide Wait. So download speed and incremental rendering were genuine issues.

6.2.1 Colgroups

See Colgroups

Here is a picture of a table with colgroups. The heavy lines show how the columns of the table have been grouped into two colgroups, with each colgroup containing two columns.



Colgroups provide a number of convenience features. The HTML specification says, for example, that authors may highlight a colgroup or rowgroup structure using style sheets or HTML attributes, and the specification provides a sample table illustrating such a use of

colgroups. An attribute setting of RULES="GROUPS" on the TABLE element would cause rules (that is, lines) to be displayed between colgroups and rowgroups rather than between cells. And so on.

These convenience features are not the reason, however, for the existence of colgroups. The original motivation for the inclusion of colgroups in the HTML specification was the desire to support incremental rendering of data. As the introduction to the section on tables in the HTML 4.01 specification says:

> The HTML table model has been designed so that, with author assistance, user agents may render tables incrementally (i.e., as table rows arrive) rather than having to wait for all the data before beginning to render.
>
> In order for a user agent to format a table in one pass, authors must tell the user agent: The number of columns in the table... [and the] widths of these columns... using a combination of COLGROUP and COL elements. If any of the columns are specified in relative or percentage terms..., authors must also specify the width of the table itself.

Normally, when a browser renders a table, it makes two passes over the table. In the first pass, it examines the entire table, discovering the largest number of cells per row in the table and how big the contents of the cells are. From this information, it calculates the widths that it will use to display the columns in the table. Then, in a second pass, it actually displays the table. But if the COL and COLGROUP elements supply the browser up front with all of the column width information, that removes the need for the browser to make the first pass. It can immediately start rendering the table. And it can do this incrementally, displaying each row as soon as it gets it.

We won't go into the details of how COL and COLGROUP specifications are coded in HTML — that material is available in the section on column groups in the HTML language specification, and we give an example of it later in this paper, in the discussion of the SCOPE example on the Access Board's Web site. But we will note that COL and COLGROUP tags must be placed in a specific location inside the TABLE element — after the CAPTION element (if present) and before the THEAD, TFOOT, and TBODY elements.

6.2.2 Rowgroups

See Rowgroups

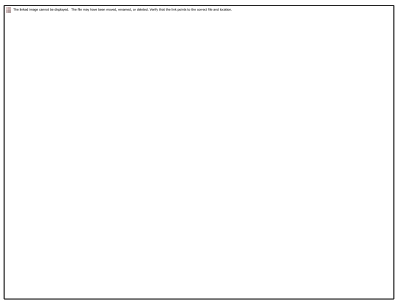
Colgroups are defined using the COLGROUP tag. Perhaps surprisingly, however, HTML contains no corresponding ROWGROUP tag. HTML has the *concept* of a rowgroup, and "rowgroup" is a valid value of the SCOPE attribute of TH and TD tags, but it contains no ROWGROUP tag, as such.

The reason that HTML has no ROWGROUP tag is that it recognizes three different kinds of rowgroups, and has specific tags for each of those kinds: THEAD, TFOOT, and TBODY. The

reason for this three-fold division is that rowgroups were designed to support a display in which the header and footer of a table are held steady, and the user can scroll through the rows in the body of the table. Here's what the HTML specification says:

> Table rows may be grouped into a table head, table foot, and one or more table body sections, using the THEAD, TFOOT and TBODY elements, respectively. This division enables user agents to support scrolling of table bodies independently of the table head and foot.

Here's a simple illustration. Suppose we had marked up our example table with THEAD, TFOOT, and TBODY tags. In the following graphic, we have colored the header row marked with the THEAD tag, and the footer row marked with the TFOOT tag. The rows in the middle have been marked with the TBODY tag. This capability would allow a browser to hold the colored rows steady, while the user scrolled through the middle rows, perhaps with a slider bar like the one in the illustration.



It was envisioned that this feature might also support the printing of tables.

> When long tables are printed, the table head and foot information may be repeated on each page that contains table data.

We won't go into the details of how THEAD, TFOOT, and TBODY specifications are coded in HTML — that material is available in the section on row groups in the HTML language specification. But we will note that you can *not* code these elements in the order that you'd expect — namely: THEAD, TBODY, TFOOT. If you code TFOOT, it must be coded before TBODY, in this order — THEAD, TFOOT, TBODY. This makes perfectly good sense when you remember the W3C's interest in supporting the incremental rendering of tables. As the HTML language specification says

> TFOOT must appear before TBODY within a TABLE definition so that user agents can render the foot before receiving all of the (potentially numerous) rows of data.

6.2.3 An Important Limitation of Colgroups and Rowgroups

It is important to understand the motivations behind HTML's colgroup and rowgroup features, because these motivations explain the characteristics and limitations of colgroups and

rowgroups. And they do have an important limitation. *They cannot be nested to more than two levels.* They support only two levels: colgroup and col, or rowgroup and row.

6.2.4 Scope

Once the W3C had colgroups and rowgroups available, they could be extended to do other work as well as the work for which they were originally intended. And so the *SCOPE* attribute was born.

The SCOPE attribute is an attribute that you can attach to header cells that are marked with TD and TH tags. It allows a header cell to be associated not just with a row or a column, but with a rowgroup or a colgroup. So the four allowable values of SCOPE are: col, colgroup, row, rowgroup.

In some ways, SCOPE is HEADERS turned inside out. That is, if you want to associate some data cell D with some header cell H, you can either code a HEADERS attribute on data cell D, or a SCOPE attribute on header cell H. The SCOPE attribute is a way for a cell to broadcast its "headerness" — its ability to function like a header cell — to other cells.

The SCOPE attribute has significant limitations.

1. The hierarchy provided by rowgroups and colgroups is limited to two levels. This limitation is a result of the original purpose and design of rowgroups and colgroups. This means that if you have a table with three or more levels of column headings, you cannot use colgroups to associate data cells with column header cells, and similarly for rows in which you have three or more levels of row headers.
2. In order to use colgroups and rowgroups for any serious table markup, it is necessary to mark up the table with COLGROUP and TBODY tags. This adds an extra layer of complexity to the table markup.

6.2.5 Confusion About the *Scope* Attribute

It is important to note that a colgroup is not defined by a *colspan* attribute.

It is a common mistake to code something like the following:

```
<TH COLSPAN="2" SCOPE= "colgroup"> Some heading text </TH>
```

In this code, the author has defined a cell that spans two positions, and had added the SCOPE specification of "colgroup" in order to make the scope of the header cover both of the spanned columns. *But this is a mistake. The COLSPAN attribute has no bearing on the effect of the SCOPE attribute.*

SCOPE seems to be widely misunderstood. The W3C's Web site containing the HTML language specification gives an example of its use that — although correct — is very misleading. And the Web site of the Access Board (which has the status of a legally binding definition of the

practices to which Federal agencies must conform in order to be 508-compliant) contains an example of the use of the SCOPE attribute that is simply wrong.

6.2.6 SCOPE Example in the HTML language specification

Misunderstanding of the use of SCOPE is practically guaranteed by an example in section 11.4.1 of the HTML definition document. That example shows the following table:

Community Courses -- Bath Autumn 1997				
Course Name	**Course Tutor**	**Summary**	**Code**	**Fee**
After the Civil War	Dr. John Wroughton	The course will examine the turbulent years in England after 1646. *6 weekly meetings starting Monday 13th October.*	H27	£32
An Introduction to Anglo-Saxon England	Mark Cottle	One day course introducing the early medieval period reconstruction the Anglo-Saxons and their society. *Saturday 18th October.*	H28	£18
The Glory that was Greece	Valerie Lorenz	Birthplace of democracy, philosophy, heartland of theater, home of argument. The Romans may have done it but the Greeks did it first. *Saturday day school 25th October 1997*	H30	£18

The HTML code accompanying the example does not define a COLGROUP anywhere, and the HTML for the first cell is this.

```
<TR> <TH colspan="5" scope="colgroup"> Community Courses — Bath Autumn 1997
                                    </TH> </TR>
```

This markup appears to be using the SCOPE="colgroup" specification without any colgroups being defined on the table — this makes it easy to jump to the conclusion that a colgroup is being defined by the `colspan="5"` specification.

What the example does not point out is that — in the absence of explicit COLGROUP tags -- all tables contain a single implicit colgroup. That is the only reason this example works — the colgroup referred to in the SCOPE attribute is the single implicit colgroup that all HTML tables have by default.

6.2.7 SCOPE Example on the Access Board Web Site

An outright mistake in the use of SCOPE can be found on the Web site for the Access Board, which contains the following explanation.

> **How can HTML tables be made readable with assistive**

technology? ...

Using the Scope Attribute The first row of each table should include column headings. Typically, these column headings are inserted in <TH> tags, although <TD> tags can also be used. These tags at the top of each column should include the following attribute:

scope="col"

By doing this simple step, the text in that cell becomes associated with every cell in that column.

At first, it looks like the Access Board has merely failed to imagine tables that contain more than one row of headers. But farther down on the Web page, we find the following example, which is clearly incorrect. (I have tweaked a few attributes on of the TABLE tag, to make the layout of the table a bit easier to see.)

```
<table BORDER="1" cellspacing="0"
cellpadding="5">

 <tr>
   <th> </th>

   <th colspan="2" scope="col" > Winter </th>
   <th colspan="2" scope="col" > Summer </th>
 </tr>

 <tr>

   <th> </th>
   <th scope="col"> Morning   </th>
   <th scope="col"> Afternoon </th>

   <th scope="col"> Morning   </th>
   <th scope="col"> Afternoon </th>
 </tr>

 <tr>

 <td scope="row"> Wilma </td>
   <td> 9-11 </td>
   <td> 12-6 </td>

   <td> 7-11 </td>
   <td> 12-3 </td>
 </tr>

 <tr>
```

```
 <td scope="row"> Fred </td>
  <td> 10-11 </td>
  <td> 12-6 </td>

  <td> 9-11 </td>
   <td> 12-5 </td>
 </tr>

 </table>
```

This table is rendered this way:

	Winter		Summer	
	Morning	**Afternoon**	**Morning**	**Afternoon**
Wilma	9-11	12-6	7-11	12-3
Fred	10-11	12-6	9-11	12-5

This coding is incorrect.

By specifying SCOPE="col" on cells (Winter, Summer) that span two columns, the Access Board example clearly shows a failure to understand the use of SCOPE. To be correct, SCOPE must be "colgroup" on the cells that span two columns, and in this table there must be three COLGROUP elements — one for the stub header column, and two for the major headings. The colgroups for the major headings must specify SPAN="2". Here is what the correct code would look like. We have underlined changes to the code and marked them red. The specification SPAN="1" on the first COLGROUP is optional, but we have coded it here to make it explicit.

```
<table BORDER="1" cellspacing="0">
<COLGROUP SPAN="1">




<COLGROUP SPAN="2">
 <COLGROUP SPAN="2">

<tr>
   <th>   </th>

   <th colspan="2"
scope="colgroup">Winter </th>
   <th colspan="2" scope="colgroup"
>Summer </th>
      </tr>

 ... etc. ...
</table>
```

6.3 The BASIC algorithm

The BASIC algorithm is described in section 11.4.3 of the W3C specification for HTML 4.01. I reproduce it here, with the steps numbered for easy reference.

> ### 11.4.3 Algorithm to find heading information
>
> In the absence of header information from either the scope or headers attribute, user agents may construct header information according to the following algorithm. The goal of the algorithm is to find an ordered list of headers. (In the following description of the algorithm the table directionality is assumed to be left-to-right.)
>
> 1. First, search left from the cell's position to find row header cells. Then search upwards to find column header cells. The search in a given direction stops when the edge of the table is reached or when a data cell is found after a header cell.
> 2. Row headers are inserted into the list in the order they appear in the table. For left-to-right tables, headers are inserted from left to right.
> 3. Column headers are inserted after row headers, in the order they appear in the table, from top to bottom.
> 4. If a header cell has the headers attribute set, then the headers referenced by this attribute are inserted into the list and the search stops for the current direction.
> 5. TD cells that set the axis attribute are also treated as header cells.

Let's see how this works in practice. Consider the following table.

Example Table 1

Ruritanian Population Survey		By Gender	
		Males	Females
By Region	North	2111	2222
	South	4444	5555

The HTML for this table is:

```
<TABLE BORDER="1" CELLSPACING="0" CELLPADDING="5">
<CAPTION>Example Table 1</CAPTION>

<TR>
<TH ROWSPAN="2" COLSPAN="2"> Ruritanian <BR> Population <BR>
Survey </TH>
<TH ROWSPAN="2"> All <BR> Genders </TH>

</TR>
<TR>
<TH> Males </TH>
<TH> Females </TH>
</TR>

<TR>
<TH ROWSPAN="2"> By Region </TH>
<TH> North </TH>
<TD ALIGN="right"> 2111 </TD>

<TD ALIGN="right"> 2222 </TD>
</TR>

<TR>
<TH> South </TH>
<TD ALIGN="right"> 4444 </TD>

<TD ALIGN="right"> 5555 </TD>
</TR>

</TABLE>
```

According to the BASIC algorithm, what is the procedure used to construct the ordered list of headers associated with the cell with value 5555 in this table? And what is the resulting list?

Note that:

- The column header cell "By Gender" has an anchor position of (1,4), but because of the attribute *colspan="2"* , it also covers position (1,5).
- The row header cell "By Region" has an anchor position of (4,1), but because of the attribute *rowspan="2"*, it also covers position (5,1).
- Both the "By Gender" and "By Region" cells cover a position that has the same row or column coordinate as the data cell, whose anchor position is (5,5).

The search starts from the anchor position of the data cell, and is carried out in the way that a sighted reader would carry it out. That is, if a row header cell covers a row that is the same as the data cell's row, then that header cell will be discovered during the search. And similarly, if a column header cell covers a column that is the same as the data cell's column, then that header cell will be discovered during the search.

Here is the search procedure.

1. We create an empty list.
2. Our starting point for the search is the anchor position of the data cell — (5,5).
3. We start searching left from the starting point, along row 5. We find header cells "South" at position (5,2) and "By Region" covering position (5,1). The search is terminated at the left edge of the table.
4. We add these headers to the list in the left-to-right order in which they appear in the table — that is, by the order of the columns in the cells' anchor positions: "By Region" and "South".
5. We start searching upward from the starting point, along column 5. We find header cells "Females" at position (2,5) and "By Gender" covering position (1,5). The search is terminated at the top edge of the table.
6. We add these headers to the list in the top-down order in which they appear in the table — that is, by the order of the rows in the cells' anchor positions: "By Gender" and "Females".

This search produces the following results list.

```
By Region
South
By Gender
Females
```

From this, we can see that the BASIC algorithm works very nicely even if the table contains multi-level column or row headers. Why the WCAG recommend the use of additional markup for such tables is a mystery.

Now lets look at a table that uses our third formatting technique: indented stubs using cascading stylesheets.*NOTE: This table is probably the most important example in this paper.*

Ruritanian Ore Production			
	1998	**1999**	**2000**
Copper Ore	123	123	123
Unrefined	123	123	123
Less than 40% copper	123	123	123
40% or more copper	123	123	123

The HTML for this table is:

```
<TABLE CELLSPACING="0" CELLPADDING="5" BORDER="1"
ALIGN="CENTER">
<CAPTION>Ruritanian Ore Production</CAPTION>
<TR>
  <TH> </TH>

  <TH>1998</TH>
  <TH>1999</TH>
  <TH>2000</TH>
</TR>

<TR>
  <TH ID="R1">Copper Ore</TH>
  <TD>123</TD>
  <TD>123</TD>

  <TD>123</TD>
</TR>
<TR>
  <TH ID="R2" HEADERS="R1"><div
class="sub1">Unrefined</div></TH>

  <TD>123</TD>
  <TD>123</TD>
  <TD>123</TD>
</TR>

<TR>
  <TH ID="R3" HEADERS="R1 R2"><div class="sub2">Less than 40%
copper</div></TH>
  <TD>123</TD>
  <TD>123</TD>

  <TD>123</TD>
</TR>
<TR>
```

```
  <TH ID="R4" HEADERS="R1 R2"><div class="sub2">40% or more
copper</div></TH>

  <TD>123</TD>
  <TD>123</TD>
  <TD>999</TD>
</TR>

</TABLE>
```

According to the BASIC algorithm, what is the procedure used to construct the ordered list of headers associated with the cell with value 999 in this table? And what is the resulting list?

The search starts from the anchor position of the data cell, and is carried out in the way that a sighted reader would carry it out. That is, if a row header cell covers a row that is the same as the data cell's row, then that header cell will be discovered during the search. And similarly, if a column header cell covers a column that is the same as the data cell's column, then that header cell will be discovered during the search.

Here is the search procedure.

1. We create an empty list.
2. Our starting point for the search is the data cell.
3. We search left until we find the header cell "40% or more copper". The search is terminated at the left edge of the table.
4. We add this header to the list.
5. Since this header cell has a HEADERS attribute, we follow the IDs in the HEADERS attribute to cells R1 and R2, add their cell contents to the list, prior to "40% or more copper", and the search for row headers is terminated.
6. We search upward until we find the find column header cell "2000", and insert it into the list. The search is terminated at the top edge of the table.

This search produces the following results list.

```
Copper Ore
Unrefined
40% or more copper
2000
```

The magic bullet here is step 4 of the BASIC algorithm.

```
.If a header cell has the headers attribute set, then the headers
referenced by this attribute are inserted into the list and the
search stops for the current direction.
```

This seems to be the simplest and most straightforward technique of all.

Perhaps it is worth pointing out one possible pitfall for the unwary. In the BASIC algorithm, when the search finds a header cell with a HEADERS attribute, after that cell is processed *the search stops.* So if we had a table that had two columns of header cells, and the inner header cells had a HEADERS attribute, this way:

```
<TR>
   <TH>Outer Header</TH>
   <TH HEADERS="R1 R2">Inner Header<TH>
   ... and some data cells here ...
<TR>
```

then the BASIC algorithm, when searching for headers, would never find the outer header.

7 Conclusion and Recommendations

Given the variety of techniques for visible and accessible formatting, which one should you choose? Here are some evaluation criteria for accessible HTML tables, with the most important first.

1. First, the HTML code must conform to a standard for accessible HTML code. The acceptability of the code is *not* judged by the success with which it can be read by any particular screen reader product, or version of screen reader product. Acceptability is judged by measurement against a standard for accessible HTML coding practices.
2. The code must produce an acceptable visual presentation as well as an accessible presentation. We cannot adopt a set of coding practices that produces tables that are acceptable to the vision-impaired members of our user communities, but unacceptable to the sighted members.
3. It must be feasible for human beings (using whatever support technology is available) to create HTML tables. Athough many statistical tables are generated from databases by software programs, many are not. It must be possible for statistical analysts to create hand-crafted tables for special purposes.
4. The markup must be supported by the current generation of screen-reader technology.
5. In order to minimize download time, compact markup is preferred to verbose markup.

One of the most important considerations is the visual appearance of statistical tables. The three visual formatting techniques that we have discussed — nested stub headers, indented stub headers using cell control, and indented stub headers using CSS control -- all produce slightly different results. If one of those visible formats is required, or acceptable, or unacceptable for your organization, then that will have a decisive effect on your decision.

In most organizations, nested stub headers are simply unacceptable. So, for most organizations, that option can be eliminated immediately.

Most statistical organizations publish a mix of tabular data. Some table are generated automatically from database data, while other tables are hand-crafted by statisticians and used in analytic reports. If your organization publishes such a mix of tables, then you will need a format that is simple enough to be hand-coded by a human being. The formats that use cell control tend to be complicated. Using ROWSPAN is very difficult. COLSPAN is less difficult, and just using empty cells (without ROWSPAN or COLSPAN) is less complicated, although still cumbersome.

In comparison, using CSS to control the stub margins is extremely simple.

Of the three accessibility techniques that we have examined — the HEADERS algorithm, the SCOPE algorithm, and the BASIC algorithm — neither the HEADERS nor the SCOPE algorithm seems very attractive. Marking up a table with HEADERS and ID attributes is time-

consuming and difficult. Using the SCOPE algorithm is easier (although not so easy that it can't be mis-understood, as we've seen), but it is impractical because it is limited to two levels of nesting. Clearly, the BASIC algorithm is the most attractive choice of the three. Marking up the stub header cells with HEADERS and ID attributes is cumbersome, but the process is relatively straightforward and for small analytic tables it is feasible for a human being. Adding such markup with software is not difficult. The code is relatively compact.

Our conclusion, therefore, is that — for most organizations, for regular tables — the best option is to use CSS control to create indented stubs, and to use the BASIC algorithm to make the tables accessible. While not as simple and easy as one might wish, it is — considering the alternatives — clearly the most attractive option available. (Click HERE to go to the example in this paper. Another example is available in the HTML Tables Cookbook.)

For irregular tables, the best option is to reformat the tables into regular tables. If that is impossible, and the table must remain irregular, then in most cases the best visible formatting technique will be to use CSS margin control, and the only possible accessible formatting technique will be the HEADERS algorithm.

8 Some Visual Formatting Issues

8.1 Fonts

Generally speaking, data cells should appear in a monospaced format such as Courier, so that columns line up nicely. On the other hand, header cells generall should appear in a proportional font, both for the sake of a nice appearance and because text in a proportional font will take up less space. Using a proportional font for your stub headers, is especially important, because long or deeply nested headers will tend to make your tables spread out horizontally, and you need to counteract that tendency as much as possible.

The easiest way to control fonts in a table is by using a CSS stylesheet to attach font specifications to TH and TD cells. Here is an example:

```
td {font-family:   Courier monospace}
th {font-family: "Times New Roman" serif proportional}
```

8.2 Horizontal Cell Alignment

Vertical alignment is specified on the ALIGN attribute of TD and TH cells, or may be controlled via CSS stylesheet settings.

Generally speaking, data cells should be right-justified, so that data columns line up nicely. Column header cells should be centered. And stub headers should be left-justified.

The easiest way to control cell alignment is with a CSS stylesheet that attaches alignment specifications to the TH and TD cells. In the following example, we have attached alignment specifications to the styles that we defined when we discussed fonts.

```
td {text-align: right; font-family:    Courier monospace}
th {text-align: left ;  font-family: "Times New Roman" serif proportional}
.center {text-align: center}
```

In this example, not that we have also defined a style for centered alignment.

The problem with attaching an alignment specification to TH cells is that, no matter what alignment you choose, it will be wrong for some of the header cells. If you choose to align header cells to the left, that will be good for stub header cells but bad for column header cells. If you choose to alight header cells to the center, then that will be good for column header cells but bad for stub header cells.

On the grounds that stub header cells are likely to be more numerous than column header cells, we have given TH cells a "left" alignment. This means, that for column header cells, we will need to over-ride this specification by an explicit CLASS attribute on the column header cells, this way.

```
<TH class="center" ... > Header Cell Text </TH>
```

The HTML specification includes an alignment option of "char" (character) that would allow you to specify a decimal delimiter character as the alignment character. Using this option, English-speaking Web masters should be able to align their decimal numbers on a period, French-speaking Web masters should be able to align their decimal numbers on a comma, and so on. However, this feature does not currently seem to be widely supported in visual browsers, so it should be avoided. If your data table contains decimal numbers, and you want to insure vertical alignment of you numbers on the decimal point, all numbers should be padded to the right of the decimal point to the length of the longest decimal. To avoid presenting a number with more precision than it actually has, the pad character should be " " (non-breaking space) rather than "0" (zero).

8.3 Vertical Cell Alignment

Vertical alignment is specified on the VALIGN attribute of TD and TH cells, or may be controlled via CSS stylesheet settings.The default cell alignment is "middle", which is virtually always wrong.

Generally speaking, the vertical alignment for header cells (both column and stub headers) should be "top". Depending on the type of contents of the data cell, the appropriate vertical alignment may be either "top" or "baseline". When data cells contain only a single number, the vertical alignment should be "baseline", so that numbers line up horizontally with the last line of the stub header. On the other hand, when the data cell may contain a multi-word note or sentence, the vertical alignment should be "top".

The *baseline* is an imaginary line on which letters rest. Some letters with "descenders" may have parts that extend below the baseline, such as the tail of "g" and "y". You can see where the baseline falls by looking at a word such as "My". The bottom of the "M" rests on the baseline, and you can see the tail of the "y" dips below the baseline. If you mix fonts of different sizes on the same line (or in the same row of a table) and they are aligned according to the bottoms of their descenders, it looks very messy. It looks much better if their baselines are aligned. This is what VALIGN="BASELINE" does. It sets the row so that all the cells share the same baseline. (See the iDocs Guide to HTML for a graphic that illustrates this situation.)

8.4 Dot Leaders

In printed tables, stub headers are often followed by a series of periods ("dot leaders") that help the user track across the page.

```
East.............  109.2
Southwest.......   99.3
All Regions.....  101.5
```

There is no way to do this in HTML. Don't even think about it. In any event, in an HTML table, most cells will be enclosed in lines, so the dot leaders are not necessary.

8.5 Footnotes

Notes, or footnotes, are often a part of of statistical data tables. Logically, the notes to a table are just as much a part of the table as, say, the table's caption Unfortunately, the HTML specification for tables makes no provision for notes, so we must cobble together note handling from other HTML features.

The general topic of "notes" actually covers two related topics: note references (the little symbols or numbers that appear in the table and refer you to a note) and the note proper (the text that the footnote reference refers to).

Footnote references need to be marked in some way so that it is possible to distinguish them from the normal cell contents. The most common practice is for note references to be superscripted numbers. If a cell has more than one reference, the references are separated by commas. You do this in HTML by enclosing the references list in <SUP> (superscript) tags.

```
<TH><sup>6, 7</sup> Header Cell Text</TH>
```

In header cells (where the cell alignment is "left" or "center"), footnote references are usually positioned after (that is, be to the right of) the cell contents. In data cells (where the cell alignment is "right"), footnote references are usually positioned before (that is, to the left of) the cell contents. Since the monospace font of data cells may make it hard to distinguish the footnote reference from the contents of the cell, it may be desirable to set off the footnote references by enclosing them in parentheses or brackets, or by putting one or two " " elements between them and the cell contents.

The footnotes are usually positioned at the bottom of the table.

The real question with regard to footnotes is: How should footnote references behave? You should make them links that jump to the corresponding footnote.

```
... <sup><a href="#table2footnote3">3</a></sup> ...
...
... <a name="table2footnote3">3.</a>

    Text of footnote 3 for table 2...
```

If a user clicks on the link and jumps too the footnote text, he can easily return to his original position in the table by clicking on his browser's BACK button (or doing the hotkey equivalent).

Footnotes can be formatted in any way that you prefer. Here is a technique that seems to produce a good appearance for footnotes.

The footnotes are placed in a borderless table, immediately following the data table. The footnote table is specified to be the same width as the data table, so the width of the footnotes matches that of the data table (in the following example, both tables WIDTH setting is "50%".) The footnote table has two columns, one for the footnote number and one for the footnote text. The cells containing the footnote numbers are horizontally aligned to the right and vertically aligned to the top. The cells containing the footnote text are horizontally aligned to the left. Here is an example.

Table With Footnotes	**1999** [1]
Copper Ore [2]	[3] 999.99

1. This is a long footnote, in order to show how the text will wrap around in the data cell of this invisible table. Note that the footnote numbers in the first column of this invisible table are specified with VALIGN="TOP" so that they line up with the top of the footnote text rather than being vertically centered against it.
2. Text of footnote 2 for table 2.
3. Text of footnote 3 for table 2.

Here is the underlying HTML.

```
<TABLE BORDER="1" CELLSPACING="0" ALIGN="center"
CELLPADDING="5"  WIDTH="50%">
  <TR>

    <TH> Table With Footnotes</TH>
    <TH class="center"> 1999
        <SUP><A HREF="#table2footnote1"> 1 </A></SUP>
```

```
    </TH>
  </TR>

  <TR>
    <TH VALIGN="baseline"> Copper Ore
        <SUP><A HREF="#table2footnote2"> 2 </A></SUP>

    </TH>
    <TD> <SUP><A HREF="#table2footnote3"> 3 </A>    
999.99</SUP></TD>

  </TR>
</TABLE>


<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="5"
ALIGN="center"  WIDTH="50%">
        <TR>
    <TD CLASS="right" VALIGN="top"><A
      NAME="table2footnote1">1.</A></TD>

    <TD CLASS="left">This is a long footnote, in order to show
how the
      text will wrap around in the data cell of this invisible
table.
      Note that the footnote numbers in the first column of
this
      invisible table are specified with VALIGN="TOP" so that
they
      line up with the top of the footnote text rather than
      being vertically centered against it.</TD>
  </TR>

  <TR>
    <TD CLASS="right" VALIGN="top"><A
NAME="table2footnote2">2.</A></TD>

    <TD CLASS="left">Text of footnote 2 for table 2. </TD>
  </TR>

  <TR>
    <TD CLASS="right" VALIGN="top"><A
NAME="table2footnote3">3.</A></TD>

    <TD CLASS="left">Text of footnote 3 for table 2.</TD>
  </TR>
</TABLE>
```

8.6 Rules

The RULES attribute on the TABLE element gives us some more options for formatting a table visually. In this table, we'll specify just RULES="ROWS".

rules="rows"				
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99

In this table, we'll specify RULES="COLS". If we specify CELLPADDING="0", then the figures in the cells will bump right up against the vertical rules separating the columns — very ugly. Here is where CSS styles can help us. We will specify that TD cells should have a left and right margin of at least 1 em: {margin-left: 1em; margin-right: 1 em}.

rules="cols"				
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99

Now lets add in some column headers.

rules="cols 2 "				
HEADER A	**Header B**		**Header C**	
	Header B1	**Header B2**	**Header C1**	**Header C2**
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99

Not very attractive. Let's try specifying RULES="GROUPS". To get a horizontal rule separating the header cells from the body cells of the table, we need to make the headers into a group. We can do this using the THEAD tag.

rules="groups 1"				
HEADER A	**Header B**		**Header C**	
	Header B1	**Header B2**	**Header C1**	**Header C2**
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99

That is about as good as we're going to be able to do with RULES.

We'd like to be able to draw borders around all of the cells in the table head, but only vertical rules between the column group, but unfortunately we can't. The problem with the RULES attribute is that you can't provide separate RULES specifications for THEAD and TBODY.

It is possible to kludge something together by playing around with inserting <HR> tags at various places. The best-looking result that I've been able to obtain was by inserting <HR> before the text in the cells for headers B1, B2, C1, and C2. The result looks like this.

rules="groups 2"				
HEADER A	**Header B**		**Header C**	
	Header B1	**Header B2**	**Header C1**	**Header C2**
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99

When the BASIC algorithm goes looking for column headers, the column header list for the rightmost column would be:

```
Header C
<HR>Header C2
```

which might be acceptable when read by a screen reader.

We can get separators between the columns (rather than the groups) if we specify RULES="COLS" and put in <HR> tags before the text in the cells for headers B1, B2, C1, and C2, and before the text in each of the cells in the first row of data cells. Here is the result.

rules="cols 3 "				
HEADER A	**Header B**		**Header C**	
	Header B1	**Header B2**	**Header C1**	**Header C2**
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99
Category X	999.99	999.99	999.99	999.99