

An Introduction to Dynamic Response Formatting

by Christopher Boudreau

A Senior Project counted as a semester's course toward a
BA degree in Computer Science at Boston College
January - April 1997

INTRODUCTION

The Concern: Dynamic Web Content from Data Sources

Since the birth of the World Wide Web, web publishers have tried to overcome the limitations of static content on web pages. The HyperText Markup Language (HTML) is inherently static; text and images are drawn to the screen and cannot be moved. The position of all elements in an HTML document* is calculated once when the page is loaded in a browser and then is not calculated again until the page is reloaded. Common web programming tools such as JavaScript, Java, ActiveX and Common Gateway Interface (CGI) programs have been developed to make web pages more dynamic and active. These tools have added animations, calculations, transactions, and dynamic content to web pages. The dynamic content has been made possible by programs (written in all of the above languages) that interface with a web server – a user executes an interface program that can display different results, depending on user input.

A major concern for information providers, publishing over the Web, has been to deliver up-to-the-minute content from data sources. Sometimes these data sources are databases, local and external. A web publisher may want to list current prices from a local database of products, for example, or show financial information on companies from an external database. One can also imagine web pages themselves being considered data sources. What if a web publisher wanted to display the current day's headlines according to *Headline News* (<http://www.headlinenews.com>)? Or to display the week's forecast from *USAToday* (<http://www.usatoday.com>) ?

The Server-Side Solution

Traditionally, CGI programs have answered this concern. A CGI program, given a user's request, can execute a query on a database and return the results in a web page. For a developer to implement this solution, however, he would need to be strongly proficient in CGI programming – the task is nontrivial. The programmer would need to be able to execute a query on the data source, convert the information returned to data types he can work with in a web page, and decide for each query response, how he would like to display the results on the page. Code would need to be rewritten if the programmer wanted to display the information in a different way or to query a database with a different structure. Such a large CGI program would also take up web server resources.

The Client-Side Solution

What if more could be done on the browser client? What if there could be a way to easily change the display of a query response without having to rewrite much code? What if there was a system already in place to query any database in the same manner and to receive responses back in the same format? What if web sites could be queried for information as if they were databases? What if the door to database driven web content could be opened up to more than just highly experienced programmers?

* following all HTML standards up through 3.0

Dynamic Response Formatting

Thanks to the work of the Context Interchange (COIN) team at the Massachusetts Institute of Technology two of these questions have already been answered. The COIN team has implemented both the ability to make a query to any database (receiving responses in a homogenous format) and the ability to request information from a website as if it were a database. This technology, along with help from Server Side Includes and JavaScript, has allowed beginner programmers to receive hassle-free, database driven web content through Dynamic Response Formatting.

The protocol of Dynamic Response Formatting consists of three parts. One, the execution of a web server program that takes as input a SQL query. This program executes the query on the data source, database or website, and returns the result in the form of a JavaScript object; the JavaScript object is written to the web page through a Server Side Include. Two, the inclusion of JavaScript libraries to display the query response in varied formats. These libraries not only include various display functions but also methods for performing calculations on the object. And three, the execution of selected JavaScript library functions to display the response on the page in the desired format.

At this time, other web programming techniques are being released to allow inexperienced programmers the ability to place dynamic database content on their web pages. Both Netscape Communications Corporation and Microsoft Corporation are developing a new form of HTML, called Dynamic HTML (D-HTML), that will allow for dynamic content on web pages. DHTML content can change its position on a web page without reloading from the server. This paper will explore the differences and similarities between Dynamic Response Formatting and DHTML.

An Example: Weather Forecasts

To fully explain how Dynamic Response Formatting works, let us go through an example. This example will show how to include dynamic weather forecasts on a web page.

Step 1: Executing the Query

The dynamic content in Dynamic Response Formatting starts with the Server-Side Include. A Server-Side Include (SSI) is an embedded command in a web page that the web server executes before delivering the page to the requesting client. Web servers set to look for SSI's will examine documents with the extension ".shtml" for which a browser is making a request. It will parse the document before delivering it to the client, looking for any SSI commands. If it finds any command, it will execute it, then send the page on to the client's browser.

One SSI command is "exec". This command can be used to execute any program located on the web server. In this example, the program being called is located on context.mit.edu.

The COIN team has written a program called "wrapper" that takes a SQL query and a name as input, executes the query on the specified source, and outputs a JavaScript object representing the query response. The wrapper can accept any SQL statement for which it has an understanding of the data source. The wrapper learns of data sources from specification files describing how information is organized in the database or on the website. The ability to query websites as databases, or "wrap" websites, has been developed by other members of the COIN team and work is now being done to automate the wrapping process. This means that the "web wrapper" will soon be able to accept a URL, automatically explore the specified location, and learn how information is organized on the website for querying purposes.

The syntax of the SSI is very important. For the exec command, your SSI must match the following pattern:

```
<!--#exec cmd="program" -->
```

Notice the '#' before 'exec' and the space after the last double quotation mark – this space **must** be present, followed by two dashes and a greater-than sign or the SSI will not work.

This wrapper call will include a SQL statement on USAToday for the weather and a name tag, "usatoday". Here is the SSI in full:

```
<!--#exec cmd="wrapper \"query=select usatoday.City, usatoday.Day, usatoday.Forecast,
usatoday.HighTF, usatoday.LowTF from usatoday where usatoday.State =
Massachusetts, usatoday.City=Boston;name=usatoday\" -->
```

Backslashes are used to escape quotation marks that would interfere with the execution of the SSI.

The SSI is placed in the following location in a HTML document:

```
<html>
<head>
<!--#exec cmd="wrapper \"query=select usatoday.City, usatoday.Day, usatoday.Forecast,
usatoday.HighTF, usatoday.LowTF from usatoday where usatoday.State =
Massachusetts, usatoday.City=Boston;name=usatoday\" -->
<title>Some Title</title>
</head>
<body>
</body>
</html>
```

The SSI will execute on the server when the page is requested and insert JavaScript in its place as described below*:

```
<html>
<head>
<SCRIPT LANGUAGE="JavaScript">
<!--
function array(){};
function reltable(Date, Atts, Cols, Rows, Data){
    this.asof=Date;
    this.header=Atts;
    this.cols=Cols;
    this.rows=Rows;
    this.content=Data;};

usatoday= new reltable("Now", "N/A", 0, 0, "N/A");
usatoday.header=new array();
usatoday.asof="Tue Jul 15 11:07:01 1997";

usatoday.header[1]="usatoday.City";
usatoday.header[2]="usatoday.Day";
usatoday.header[3]="usatoday.Forecast";
usatoday.header[4]="usatoday.HighTF";
usatoday.header[5]="usatoday.LowTF";
usatoday.cols=5;
```

* the query that created this response was executed Tuesday, July 15, 1997

```
usatoday.content=new array();
usatoday.content[1]=new array();

usatoday.content[1][1]="Boston";
usatoday.content[1][2]="TUESDAY";
usatoday.content[1][3]=" There will be partly cloudy skies with periods of sunshine";
usatoday.content[1][4]="79";
usatoday.content[1][5]="68";
usatoday.content[2]=new array();

usatoday.content[2][1]="Boston";
usatoday.content[2][2]="WEDNESDAY";
usatoday.content[2][3]=" Variable clouds with showers and thunderstorms";
usatoday.content[2][4]="90";
usatoday.content[2][5]="72";
usatoday.content[3]=new array();

usatoday.content[3][1]="Boston";
usatoday.content[3][2]="THURSDAY";
usatoday.content[3][3]=" There will be partly cloudy skies with periods of sunshine";
usatoday.content[3][4]="89";
usatoday.content[3][5]="69";
usatoday.content[4]=new array();

usatoday.content[4][1]="Boston";
usatoday.content[4][2]="FRIDAY";
usatoday.content[4][3]=" There will be partly cloudy skies with periods of sunshine";
usatoday.content[4][4]="87";
usatoday.content[4][5]="68";
usatoday.content[5]=new array();

usatoday.content[5][1]="Boston";
usatoday.content[5][2]="SATURDAY";
usatoday.content[5][3]=" There will be partly cloudy skies with periods of sunshine";
usatoday.content[5][4]="86";
usatoday.content[5][5]="67";

usatoday.rows=5;

// -->

</SCRIPT>
<title>Some Title</title>
</head>
<body>
</body></html>
```

This JavaScript code will create a new object called “usatoday” and fill its attribute ‘content’ with the results of the query.

The data sources that are available through the wrapper program can be viewed in Appendix A.

Step 2: Including the Libraries

The next step is to include the necessary JavaScript libraries in order to take advantage of a large series of prewritten display and formatting functions. This is also done with a SSI. The SSI command is “include”.

The full SSI is as follows:

```
<!--#include file="file.js" -->
```

And would be placed in HTML as shown below:

```
<html>
<head>
<!--#exec cmd="wrapper \"query=select usatoday.City, usatoday.Day, usatoday.Forecast,
usatoday.HighTF, usatoday.LowTF from usatoday where usatoday.State =
Massachusetts, usatoday.City=Boston;name=usatoday\" -->
<!--#include file="file1.js" -->
<!--#include file="file2.js" -->
<title>Some Title</title>
</head>
<body>
</body>
</html>
```

Each library file must have its own include statement.

Below is a description of the available JavaScript library files:

1. Table.js

This file contains all necessary JavaScript functions to display the query in an HTML table.

The functions located in this file were written with the idea that publishers would want to save certain formatting changes and yet still be able to access the original, unformatted data. The functions are written in an object-oriented manner – a table is an object that has attributes and methods. With this in mind, the library was written for developers to create copy instances of the query response and add formatting changes to those copies, instead of the original query response object. Formatting changes are saved in the duplicate table. Developers can easily use the same table object somewhere else on the page without having to redo all the formatting changes. Multiple tables consisting of the same query response can be easily made without having to undo formatting changes.

2. Format.js

This file contains functions to display the query results in any HTML style. It includes functions to change font attributes, header attributes, and basic formatting features such as italicize, bold, center, and print lists.

These functions were written to easily save formatting changes. This feature comes from the fact that an HTML tag will work around text as long as there is a start and end tag. You can make text bold, for example, and then underline it by putting the underline start and end tags (<U>,</U>) around already bolded text (text → <U>text</U>).

3. Math.js

This file contains functions to perform basic math calculations on arrays of numbers. Available functions include sum, min, max, and average.

4. Url.js

This file contains one function to add a hyperlink and target to a string.

5. Form.js

This file contains functions to display the query response in any HTML form element.

In this example, all libraries will be included to demonstrate features from each.

Below is our web page code so far:

```
<html>
<head>
<!--#exec cmd="wrapper \"query=select usatoday.City, usatoday.Day,
usatoday.Forecast, usatoday.HighTF, usatoday.LowTF from usatoday where
usatoday.State = Massachusetts, usatoday.City=Boston;name=usatoday\" -->
<!--#include file="table.js" -->
<!--#include file="format.js" -->
<!--#include file="form.js" -->
<!--#include file="math.js" -->
<!--#include file="url.js" -->
<title>Your Weather Forecast</title>
</head>
<body>
</body>
</html>
```

Step 3: Formatting the Response

With the necessary JavaScript libraries loaded, we will now be able to call available functions to display our results. Besides calling library functions, web authors comfortable with JavaScript can also write their own code to display the results in any manner not provided by the libraries.

Full library documentation is available in Appendix B. Below are the function calls that will be used in this example, correctly placed in the HTML. An explanation of what each function call does will follow the page code. This page can viewed online at <http://context.mit.edu/Steph/home/html/WS/htmldocs/weather.shtml>.

1. <html>
2. <head>
3. <!--#exec cmd="wrapper \"query=select usatoday.City, usatoday.Day, usatoday.Forecast, usatoday.HighTF, usatoday.LowTF from usatoday where usatoday.State = Massachusetts, usatoday.City=Boston;name=usatoday\" -->
4. <!--#include file="table.js" -->
5. <!--#include file="format.js" -->
6. <!--#include file="form.js" -->
7. <!--#include file="math.js" -->
8. <!--#include file="url.js" -->
9. <title>Boston Weather Forecast</title>
10. </head>
11. <body bgcolor="#FFFFFF">
12.
13.
14. <center>
15. <h2>Boston Weather Forecast</h2>
16. from USAToday.com
17. <p>
18.
19.
20. 5-Day Forecast
21.
22.
23.

24. <i>Farenheight</i>
25. </center>
26. <br clear=all>
27. <hr>
28. <TABLE BORDER=0 width="100%">
29. <script language="Javascript">
30. function changeFields(index) {
31. document.chooseday.forecast.value=usatoday.content[index][3];

```

32. document.chooseday.high.value=usatoday.content[index][4];
33. document.chooseday.low.value=usatoday.content[index][5];
34. }

35. startDataCell("left","top",1,1,0);

36. usatoday.header[1]="City";
37. usatoday.header[2]="Day";
38. usatoday.header[3]="Forecast";
39. usatoday.header[4]="High";
40. usatoday.header[5]="Low";

41. //create a copy of the query results in a Table object
42. USATodayTable=createFormatTable(usatoday);

43. //collect the weekly high temperatures
44. weekhightemps=USATodayTable.collectCol(4);

45. //collect the weekly low temperatures
46. weeklowtemps=USATodayTable.collectCol(5);

47. //calculate the max temperature
48. max_weektemp=max(weekhightemps,USATodayTable.numRows);

49. //calculate the min temperature
50. min_weektemp=min(weeklowtemps,USATodayTable.numRows);

51. //calculate the average temperature
52. max_avgtemp=average(weekhightemps,USATodayTable.numRows);
53. min_avgtemp=average(weeklowtemps,USATodayTable.numRows);
54. avg_weektemp=(max_avgtemp+min_avgtemp)/2;

55. //make it a hyperlink to its forecast below
56. max_weektemp=addURL(max_weektemp,"#table","_self");
57. min_weektemp=addURL(min_weektemp,"#table","_self");

58. document.write(fontSize("High: "+max_weektemp,"+1"));
59. br();
60. document.write(fontSize("Low: "+min_weektemp,"+1"));
61. br();
62. document.write(fontSize("Average: "+avg_weektemp,"+1"));
63. p();

64. endDataCell();
65. startDataCell("right","top",1,1,0);

```

```

66. startForm("chooseday","GET","", "self");
67. document.write(bold("Forecast"));
68. br();
69. printTextInput("forecast","",50,50);
70. br();
71. document.write(bold("High"));
72. br();
73. printTextInput("high","",3,3);
74. br();
75. document.write(bold("Low"));
76. br();
77. printTextInput("low","",3,3);

78. p();

79. days=USATodayTable.collectCol(2);
80. startFontFace("Helvetica");
81. for (i=1;i<=USATodayTable.numRows;++i) {
82. document.write(days[i]);
83. document.write("<INPUT TYPE=RADIO NAME=\"day\" VALUE=\"+i+\"
    onClick=\"changeFields(\"+i+\");\">");
84. }
85. endFont();
86. endForm();

87. endDataCell();
88. endTable();

89. p();

90. document.write("<HR>");
91. document.write("<A NAME=table>");
92. document.write(header("Query Results in Table",3));
93. USATodayTable.printVert(1,1,"100%",1,"","");
94. </script>
95. </body>
96. </html>

```

Lines 3 through 8 are our necessary Server Side Includes.

Lines 9 through 28 are basic HTML tags to print our title. We start a HTML table on line 28. This table allows us to put the weekly High, Low, and Average on the left-hand side of the page and the interactive day's forecast on the right-hand side.

At line 29, we start JavaScript code which, you may notice, takes up the majority of the page. The first thing to come is a function to update the 'Forecast', 'High', and 'Low' fields coming later in the document. This function is activated when a user clicks on one of the radio buttons as we will see later.

Line 35 is a library call to start a new data cell in a table, a function from `table.js`.

We then reassign the header array of `usatoday` to remove the beginning 'usatoday.' which came from the query response.

Lines 41 through 54 make the necessary library calls to calculate the high, low, and average temperatures for the next 5 days using our `math.js` library file.

Lines 55 through 57 use the `addURL()` function from `url.js` to hyperlink the results to reference their full day's description below.

We then use library functions to write these values to the screen in lines 58 through 63. `FontSize()`, `br()`, and `p()` are functions from `format.js`.

We start a new data cell on line 65.

In lines 66 through 78 we start a form in which we place the Forecast, High, and Low fields. Here we use functions from our `form.js` library file.

From lines 79 through 85, we build the list of available days from the query result and place a radio button next to each. As part of each radio button tag, we have a `onClick` event handler which executes the JavaScript `changeFields()` function each time a radio button is clicked. The `changeFields()` function is not part of the available libraries. It was written for this demonstration.

We then close our form and close our table.

In lines 90 through 97, we print a horizontal rule and place the query results in a table at the bottom of the page to demonstrate how easy it is to print the results in a table with the `table.js` library file.

The following page is a print out of this document as viewed in a web browser.

Comparison to DHTML

Introduction

A comparison of Dynamic Response Formatting (DRF) and Dynamic HTML (DHTML) is somewhat a comparison of apples and oranges. DHTML, under development by Microsoft and Netscape, is an enhancement to the HTML standard. DRF is a protocol in which to retrieve information from data sources and to display the information in HTML.

Dynamic HTML

Dynamic HTML is a term signifying the work of Netscape and Microsoft to develop a more dynamic standard for HTML. Dynamic HTML is a loose term because Netscape and Microsoft are developing separate technologies (under the same name) and no final standard or recommendation has been released by the W3C. Both Netscape and Microsoft are working to enhance the browser object model through scripting languages such as JavaScript and VBScript to allow scripts access to all parts of the browser document. Both are also working on taking advantage of the most recent recommendation of the W3C for cascading style sheets. Cascading style sheets will bring greater flexibility in text styles and layout. With the introduction of a new LAYER tag, absolute positioning will be available, allowing web publishers to specify x, y, and z coordinates for any portion of their document. With this new positioning ability and the ability to control all objects in a document, interactivity will be available as it never before has on the Web. Scripts will be able to hide and show parts of a document, perform sprite-like animations, and read the contents of a document on-the-fly to produce other documents, such as a table of contents. Only Microsoft has mentioned using the new features above to deliver dynamic database content on web pages.

Dynamic Response Formatting

Dynamic Response Formatting, on the other hand, is a working, Web implementation of the Context Interchange Project at MIT. The ability to query any database or website for which specification files have been constructed is a technology that has been developed by the Context Interchange team. Dynamic Response Formatting brings the responses of such queries to the Web with JavaScript. A JavaScript object is created with the query response and JavaScript libraries are used to easily display the results in many different formats on a web page.

The Similarities

The idea of holding information in objects created by scripting languages is similar to both DHTML and DRF. Since web page scripts have the ability to change the display of the page they are in, a publisher can choose portions of information to show at any time. With DHTML, scripts will allow users to change their display without reloading the document. This is not possible with the current version of JavaScript and therefore, not possible with DRF. The present version of JavaScript can only restructure a web page when it is being loaded. DHTML has the ability to restructure on-the-fly because of the new layering options under development – portions of a document can be hidden or shown without having to reconnect to the server. With the present version of JavaScript, a

page would need to be reloaded to be displayed in a different format. If DRF was performed inside a Java applet (an idea taken into consideration during development of DRF) the format of the page could be changed without reconnecting to the server.

Open database connectivity is also similar between DRF and Microsoft's proposal for DHTML. In DRF, any data source, database or website, defined by specification files, can be queried for information. These specification files are currently developed manually by examining the structure of the database or website. Work is presently under way to automate the generation of these spec files. Once these files have been created, information can be retrieved from the data source through simple SQL statements. Microsoft's proposal for DHTML includes the ability to query comma-delimited text files and databases accessible through the ODBC and JDBC protocols. By executing a query when the page is requested, Microsoft's DHTML, like DRF, can deliver dynamic content from databases – the data displayed on the web page is as current as the data source. With Microsoft's technology, a user would not have to wait for the full query response to be received before viewing the page in his browser – in DRF, the user does have to wait. DRF, unlike Microsoft's DHTML, can also query information from websites. Microsoft seems to have left the delivery of live website content up to the website publishers with Microsoft's available push technologies. In the Microsoft world, publishers can create "desktop components" in HTML that users can place on their desktops – desktop components are refreshed at intervals defined by the user.

The Difference

The main difference between DHTML and DRF is their objectives. The objective of DHTML is to enhance HTML, incorporating features such as absolute positioning, cascading style sheets, and a more comprehensive browser object model. The objective of DRF is to deliver up-to-the-minute content from databases and websites. Only Microsoft, in their DHTML proposal, has mentioned using SQL queries in web pages to deliver the same up-to-the-minute content. Microsoft's technology, however, cannot query websites for information; it can only query databases and comma-delimited text files.

The Usefulness

While reviewing DHTML and DRF, it is helpful to ask, "What is the usefulness of these technologies?" The standard of DHTML has not been agreed upon yet. Microsoft and Netscape are developing separate recommendations for the W3C that are currently incompatible with one another. If one is developing for Netscape users, one can write DHTML for the final release of Netscape Communicator but the same code will not work in any versions of Microsoft's Internet Explorer. If one is developing for Microsoft users, one can write DHTML for Microsoft's current beta of Internet Explorer 4.0 Platform Preview Release 2. Since Internet Explorer 4.0 is still in beta, however, one cannot be guaranteed that the DHTML one writes will work in the final release and of course, DHTML for Internet Explorer does not work in any version of Netscape Navigator.

DRF, on the other hand, has been designed to work for all browsers that can interpret JavaScript. Microsoft and Netscape browsers, unfortunately, do not recognize the same

set of JavaScript commands. The JavaScript of DRF has been written to fall in the intersection of JavaScript available libraries in the Netscape and Microsoft browsers. In this way, a web page will be displayed correctly no matter if it is viewed in Internet Explorer or Netscape Navigator.

Conclusion

In conclusion, Dynamic Response Formatting is a working technology web developers can use to incorporate selective database and web page content on their sites. At present, no other such tool, that allows web page designers to easily provide personalized content from data sources, exists. The Dynamic HTML standard is still under development and Microsoft, the only one to claim to provide DHTML features similar to those of DRF, only has a beta, DHTML browser in release.

Dynamic Response Formatting has been designed to be platform-independent and easy to use. By building on the intersection of JavaScript libraries, DRF is compatible with both Internet Explorer 3.x browsers (and above) and Netscape Navigator 3.x browsers (and above). Web page developers can provide content from databases and websites with only a beginner's knowledge of JavaScript – no extensive CGI programming experience is necessary. Dynamic Response Formatting allows web developers to provide live content from data sources like never before.

Until the DHTML standard is finalized, Dynamic Response Formatting is a realistic way for inexperienced programmers to quickly publish database and website content on the Web. Within the coming year, however, when Netscape, Microsoft, and the W3C come closer to agreement, DHTML will most likely over shadow DRF in its efficiency and speed. Although no one can foresee what the DHTML standard will be, it is certain that Netscape and Microsoft will strive to greatly expand the capabilities of HTML. With the growing number of data sources on the Web, it is almost certain that one of the new features will be live database content.

Appendix A

Domain	Available Fields
Baxter	Baxter.Num, Baxter.NSN, Baxter.Man, Baxter.DistNum, Baxter.Desc, Baxter.UnitofMan, Baxter.UnitofSale, Baxter.SaleManRatio, Baxter.Count, Baxter.Price
cnn	cnn.Title
dotscientific	dotscientific.Name, dotscientific.Keywords, dotscientific.CatNum, dotscientific.Spec, dotscientific.Price
finance	finance.Ticker, finance.News, finance.Time, finance.Date
fr3	fr3.Title
h_amsterdam	h_amsterdam.Name, h_amsterdam.Single, h_amsterdam.Double
headline	headline.Title
hotelg	hotelg.Country, hotelg.City, hotelg.Currency, hotelg.Name, hotelg.Single, hotelg.Double
img	img.link
imgfr3	imgfr3.link
mit	mit.coursenum
movie	movie.name, movie.genre, movie.studio, movie.phase, movie.stars, movie.producers
nyse	nyse.Ticker, nyse.Cname, nyse.Sector, nyse.Code, nyse.Type, nyse.ShareOut, nyse.Rec, nyse.Last
nyse1	nyse1.Ticker, nyse1.Cname, nyse1.Sector, nyse1.Code, nyse1.Type, nyse1.ShareOut, nyse1.Rec, nyse1.Last
olsen	olsen.Exchanged, olsen.Expressed, olsen.Date, olsen.Rate
olsen_s	olsen_s.Exchanged, olsen_s.Expressed, olsen_s.Rate
simplenyse	simplenyse.Ticker, simplenyse.Code, simplenyse.Sector, simplenyse.Cname, simplenyse.ShareOut, simplenyse.Type
t1	t1.Circ t1.CircCode, t1.Num t1.Elu t1.Cand, t1.Part, t1.Score t1.Perc
test	test.Ticker, test.Code, test.Sector, test.Cname, test.ShareOut, test.Type
travel	travel.Title
usatoday	usatoday.City, usatoday.State, usatoday.Statelink, usatoday.Day, usatoday.Forecast, usatoday.LowTF, usatoday.HighTF
ytravel	ytravel.Title
zacks	zacks.Ticker , zacks.Cname, zacks.Rec, zacks.Last

Appendix B: Library Documentation

Form.js

displayButton(name,value)

Inside an HTML form, prints a button with specified name and value. The label of the button will be its value parameter.

displayInputImage(name,imageFile,height,width)

Inside an HTML form, displays an image that acts like a submit button. “imageFile” is the path to the image and its dimensions can be adjusted by “width” and “height”.

displaySubmitButton(value)

Inside an HTML form, displays a submit button. The label of the button will be its value parameter.

endForm()

Prints “</FORM>”

endSelectList()

Prints “</SELECT>”

endTextArea()

Prints “</TEXTAREA>”

option(text)

Inside a select list in an HTML form, prints an option. The string passed as “text” will be the option’s label and value.

printCheckBox(name,value)

Inside an HTML form, prints a checkbox.

printCheckedCheckBox(name,value)

Inside an HTML form, prints a checkbox that is checked by default.

printCheckedRadioButton(name,value)

Inside an HTML form, prints a radio button that is checked by default. Only one radio button (of a series of radio buttons with the same name) can be checked at a time.

printRadioButton(name,value)

Inside an HTML form, prints a radio button. Only one radio button (of a series of radio buttons with the same name) can be checked at a time.

`printSelectList(name,textArray,arraySize,numSelected)`

Inside an HTML form, prints a select list. The “textArray” are the options of the select list, which act as their labels and values. The “arraySize” is the size of the textArray being passed. And “numSelected” is the option selected by default. To choose no option as the default, set “numSelected” equal to 0.

`printTextArea(name,text,numCols,numRows,wrap)`

Inside an HTML form, prints a textarea. “text” is the default value of the textarea. The available values for “wrap” are “off”, “virtual”, or “physical”.

`printTextInput(name,value,size,max)`

Inside an HTML form, prints a textfield.

`selectedOption(text)`

Inside a select list in an HTML form, prints an option that is selected by default.

`startForm(name,method,action,target)`

Starts an HTML form.

`startMultipleSL(name)`

Inside an HTML form, starts a multiple select list. A multiple select list is a list in which more than one element can be chosen at a time.

`startSelectList(name)`

Inside an HTML form, starts a select list.

`startTextArea(name,numCols,numRows,wrap)`

Inside an HTML form, starts a textarea. The available values for “wrap” are “off”, “virtual”, or “physical”.

Format.js

br()

Prints “
”

bold(text)

Returns bolded string “text”

center(text)

Returns centered string “text”

endB()

Prints “”

endCenter()

Prints “</CENTER>”

endFont()

Prints “”

endHeader(n)

Ends an H tag. The “n” must match the “n” value of the startHeader.

endI()

Prints “</I>”

endU()

Prints “</U>”

fontColor(text,color)

Returns string “text” set to color “color”

fontFace(text,face)

Returns string “text” set to face (font type) “face”

fontSize(text,size)

Returns string “text” set to font size “size”

header(text,n)

Returns string “text” set to header size H“n”

italicize(text)

Returns string “text” italicized

p()

Prints "<P>"

printList(textArray,arraySize,sep)

Prints the elements of "textArray" separated by "sep". "arraySize" is the number of elements in "textArray".

printOL(textArray,arraySize)

Prints an HTML ordered list

printUL(textArray,arraySize)

Prints an HTML unordered list

startB()

Prints ""

startCenter()

Prints "<CENTER>"

startFontColor(color)

Prints ""

startFontFace(face)

Prints ""

startFontSize(size)

Prints ""

startHeader(n)

Prints "<H'n'>"

startI()

Prints "<I>"

startU()

Prints "<U>"

underline(text)

Returns string "text" underlined

Math.js

`average(numArray,arraySize)`

Returns the average of the numbers in “numArray”

`max(numArray,arraySize)`

Returns the maximum value in “numArray”

`min(numArray,arraySize)`

Returns the minimum value in “numArray”

`sum(numArray,arraySize)`

Returns the summation of numbers in “numArray”

Table.js

Table.js is a little different than the other JavaScript libraries because it holds a class definition. When a page is loaded, query responses are held in “retable” objects. The function “createFormatTable(retable)” takes a retable as input and outputs a new “formatTable” object. This formatTable is a copy of the retable and will be the one to which any editing changes are made. The original retable is not changed so that a user can display the query response in many different ways on the same page. A user can refresh a table to retrieve unformatted data from a “retable” or create a new copy with the constructor, createFormatTable(retable).

Class

formatTable

Attributes:

headers - *array of row headers*
numCols - *number of columns in table*
numRows - *number of rows in table*
content - *two dimensional array holding table contents*

Methods:

refreshCell (row,col) – *resets cell to original retable cell*
refreshRow (row) – *resets row to original retable row*
refreshTable () – *resets table to original retable*
printVert (cellspacing,cellpadding,width,border,align,valign) – *prints table vertically with specified characteristics*
printHorz (cellspacing, cellpadding,width,border,align,valign) – *prints table horizontally with specified characteristics*
collectCol (colNum) – *returns array of the contents of the specified table column*

General Table.js Functions

createFormatTable(retable)

Accepts a retable object as input and outputs a new formatTable

endDataCell()

Prints “</TD>”
endRow()
Prints “</TR>”
endTable()
Prints “</TABLE>”

printColData(textArray,arraySize,align,valign)

Inside an HTML table, prints a column. The elements of the column are the elements of “textArray”.

`printDataCell(text,align,valign,colspan,rowspan,nowrap)`

Inside an HTML table, prints a cell with the value “text”. “nowrap” can equal 1 (the cell will not wrap) or 0 (the cell can wrap).

`printDataRow(textArray,arraySize,align,valign)`

Inside an HTML table, prints a row. The elements of the row are the elements of “textArray”.

`printHeaderCell(text,align,valign,colspan,rowspan,nowrap)`

Inside an HTML table, prints a header cell with value “text”. “nowrap” can equal 1 (the cell will not wrap) or 0 (the cell can wrap).

`printHeaderRow(textArray,arraySize,align,valign)`

Inside an HTML table, prints a row of header cells. The elements of the row are the elements of “textArray”.

`startDataCell(align,valign,colspan,rowspan,nowrap)`

Prints “<TD ALIGN='align' VALIGN='valign' COLSPAN='colspan' ROWSPAN='rowspan'>”. “nowrap” can equal 1 (the cell will not wrap) or 0 (the cell can wrap).

`startRow(align,valign)`

Prints “<TR ALIGN='align' VALIGN='valign'>”

`startTable(cellspacing,cellpadding,width,border,align,valign)`

Prints “<TABLE CELLSPACING='cellspacing' CELLPADDING='cellpadding' WIDTH='width' BORDER='border' ALIGN='align' VALIGN='valign'>”

Url.js

`addURL(text,URL,target)`

Returns string “text” hyperlinked to location “URL”, directed to window “target”

Contributions

Dynamic Response Formatting has been made possible by the work of many involved with the Context Interchange Project at the Sloan School of Management at the Massachusetts Institute of Technology. Stéphane Bressan, Research Scientist, played a critical role in the development of the “wrapper” program that executes queries. He also was the one to first “wrap” many of the web sites to allow such variety in web data sources. Dr. Michael Siegel and Dr. Stuart Madnick, the directors of the Context Interchange Project, gave direction to Dynamic Response Formatting and oversaw its completion. I, Christopher Boudreau, worked with the Context Interchange team from January to May 1997 as my senior final project, part of my computer science Bachelor’s Degree at Boston College.