# A Web-Based Approach to Interactive Visualization in Context


Audris Mockus and Stacie Hibino
Bell Labs, Lucent Technologies
263 Shuman Boulevard
Naperville, IL 60566 USA
{audris, hibino}@research.bell-labs.com

Todd Graves
Statistical Sciences Group
MS F600, Los Alamos National Laboratory
Los Alamos, NM 87545 USA
tgraves@lanl.gov



## ABSTRACT
This paper proposes a framework for easily integrating and controlling information visualization (infoVis) components within web pages to create powerful interactive "live" documents, or LiveDocs. The framework includes a set of infoVis components which can be placed and linked within a standard HTML document, initialized to focus on key analysis results, and directly manipulated by readers to explore and analyze data further. In addition, authors can script the manipulation of views at the user interaction level (e.g., to set view options, select items within a view, or animate a view). We illustrate our approach with a sample analysis of a real-life data set.



## Keywords
web-based information visualization, live documents, authoring visualization web pages


## 1. INTRODUCTION
Advances in Java and WWW browsers are making it possible for web-based information visualization (infoVis) to become a reality. Today, it is practical for WWW versions of scientific publications to allow their readers to *interact with*, rather than just review, visualizations of data analysis results. Such interactive documents can present graphical results *in context* as in a static, hardcopy publication while at the same time providing an interface for directly accessing and analyzing the data first-hand. In this way, readers can confirm or disprove the author's results as well as explore the data in search of additional insights. We refer to this type of interactive document with embedded, contextual information visualization components as a *Live Document* [4], or LiveDoc, for short. In this paper, we present a new set of LiveDoc principles that, based on our experience, make them effective presentation tools. We also provide technical details on the authoring of LiveDocs.

Two primary obstacles stand in the way of the realization of the LiveDoc paradigm. First, LiveDocs are far harder to compose than traditional static documents: most prospective LiveDoc authors would need to learn a new programming language and design a user interface through which their audiences can manipulate the data, when they would prefer to focus on the research content of their documents. Second, some LiveDoc readers will have neither the time for, nor the interest in, the deeper explorations that the interactivity allows. The LiveDoc author risks losing this audience altogether, particularly if the user interface requires too much time to learn, if it takes too long to access the data, or if the LiveDoc seems too radical a departure from their static document expectations. What is required is a way to aid authors in *efficiently* creating *more effective* interactive documents.

One approach to authoring online infoVis documents involves using (or, more likely, re-implementing) existing visualization technology. Although progress is still needed to increase the efficiency and scalability of infoVis on the WWW, several web-based infoVis applets and applications are emerging for analyzing data such as up-to-date financial or geographical data online (e.g., [11], [9]). Currently, however, most web-based interactive visualizations focus more on sophisticated, domain-specific views and several appear more like stand-alone applications that happen to be accessible through a web browser. Thus, such visualizations can be limited to their own domains, and they may potentially force users to experience long delays downloading complex visualizations. In addition, such views are not designed for distribution in a static form, and they may require users to spend a fair amount of time learning the system before the users can start to gain insight from their data.

In designing our approach to LiveDocs, we address these obstacles by using simple, flexible data visualization components which are easily embedded in standard web documents. In addition, we increase the analytic power of these simple components by allowing authors to easily link them together. Currently available sample components include bar charts, smoothed histograms, and dynamic tables (a form of Rao and Card's Table Lens [10], an enhanced spreadsheet-like view).

Our flexible and simple components greatly reduce readers' learning time required for interacting with the views, especially when authors use HTML controls to automate recommended tasks. The domain-independent nature of our views lets authors use them in different contexts and allows readers to transfer their learning about the views to subsequent LiveDocs. Our LiveDoc framework provides a tailorable (see, for example, [5], [2]) and in-context user interface where only the functionality pertinent to


Published in *Advanced Visual Interfaces (AVI'2000) Conference Proceedings* (pp. 181-188).

the presentation is exposed to the user. The conventional interface in the form of HTML links or HTML form widgets is provided in the appropriate location in the document, along with instructions and suggestions for their use. Placing interactive tools within the text is in the spirit of the concept of illustrations appearing in context, exemplified by [12]. Another advantage is that readers can easily and fruitfully read our documents as if they were ordinary static documents, since our components can be set to appropriate initial states to appear just like illustrations and tables in an ordinary document. Finally, despite the simplicity of individual views, they can be easily linked together to achieve substantial analytic power, see, for example [13].

Together with these advantages, our approach also allows for *efficient authoring* of online documents: web page authors can easily embed powerful visual presentations within the context of their documents through a standard applet interface. Key features for authors include the following:

- authors can add views to a web page using a simple procedure;
- authors can add links or controls for manipulating the views (e.g., selecting subsets, setting sort order);
- authors can configure options for the views, including data used, which variables to display, and an initial state for each view;
- authors can easily link views together while the system automatically takes care of technical details such as sharing data.

**Overview.** In Section 2, we describe LiveDoc benefits to the reader, followed by a discussion of benefits to the author in Section 3. We then present details about the core LiveDoc components in Section 4. In Section 5, we provide a sample scenario using LiveDocs to present some results from the analysis of some sports data on truck racing. In Section 6, we discuss some related work and in Section 7, we summarize our conclusions and describe some future work. Finally, we include technical details about authoring LiveDocs in the Appendix.

## 2. AN EFFECTIVE, APPROACHABLE USER INTERFACE FOR READING LIVEDOCS

In this section, we review LiveDoc readers' needs as summarized in the introduction and present our approach to addressing these needs.

### 2.1 Easy Access to Key Results in Context

Ideally, readers should be able to review the core content and results presented in a LiveDoc with very little, if any, more effort than that required to read a static document. In our experience of writing LiveDocs to be viewed by users who are not experts in visualization techniques, we have found this to be especially true. Most such users tend to be more interested in seeing the key results upfront rather than having to learn how to explore in order to get to the desired findings. In our approach, we address these needs through applet initialization parameters and scriptable user interactions.

The applet initialization parameters allow authors to set the initial state of a view rather than presenting a view in some default state. For example, rather than presenting a set of linked views in a default state where all data items are selected, we may set the initial state to highlight a data subset of interest. In this way, the initial state can be used to automatically present and emphasize an interesting result upfront, without requiring interactions from the user.

Scriptable user interactions allow authors to provide simple links or control widgets to the readers. Such scripts have the following key advantages:

- they provide readers with quick access to other states of a set of views, thus enabling them to focus on various results and perspectives that a set of views can provide rather than on the mechanisms on how to get to a particular state,
- they free authors from having to explain how to accomplish various user interactions,
- they enable authors to provide a series of user interactions in a single script.

For example, consider the case where the author wants readers to sort a bar chart by size and select the top three bars in the view. If authors could not script user interactions, they might need to include instructions such as "To select the top three bars, first click the right mouse button in the bar chart to access the submenu and select 'Sort by Count' to order the bar chart by height. Then, use the left mouse button to select the top three bars in the chart." Through scripting, authors can reduce their text to "Looking at the top three bars, we see that…." In this latter case, "top three bars" is a link (href in HTML lingo) to the script for sorting the bar chart and selecting the top three bars. Note how the scriptable version provides access to user interactions *in context*.

### 2.2 Simple Views and Reduced Wait Time

If readers are faced with unbearable wait times or overwhelmed with overly complex interfaces, they are less likely to adopt the LiveDoc approach to interactive visualization in context. We reduce the overhead cost of accessing the online document and related data by focusing on simple, smaller views. We also only download the data required for the specified view rather than the whole data set. We reduce readers' learning time for interacting with views by avoiding overly complex views, again focusing on simple and familiar views, and also providing scriptable links and control widgets that authors can set in context.

### 2.3 Interactively Exploring Results

The real power of a LiveDoc, however, is to go beyond static documents and support users in exploring the underlying data on their own. We improve the analytic power of the online document by providing a framework in which authors can compose a presentation of results from a set of views (i.e., select which ones they want) and easily link these views together (views are linked if user's selection in one view automatically propagates the corresponding selection to all other linked views.) Even a simple bar chart view and dynamic table, when linked together or to other views, can add power to a presentation (e.g., see Section 5.1).

Interested readers may explore the data using all of the features of the linked views to conduct their own independent investigations of the presented data and, possibly, arrive at their own conclusions that are more relevant to the reader and may be different from the ones presented by the authors.

## 3. AN EFFICIENT, EXTENSIBLE FRAMEWORK FOR AUTHORING LIVEDOCS

### 3.1 Linkable InfoVis View Components

Our framework for creating LiveDocs provides authors with a set of simple, configurable, and linkable infoVis views which authors can easily add to a web page through a standard applet interface. This approach relieves authors from the burden of programming each view by hand and supports them in creating LiveDocs with some of the effective and accessible features described in the previous section. Authors can simply pick and choose the appropriate views for their data and tailor them through author-configurable options to meet their needs.

### 3.2 JavaScript Library of Common Functions

In addition to linking, we provide a public command-type interface to the views to emulate all GUI interactions. This interface may be used to script the initial state of the views or to provide alternatives to a GUI, e.g., speech interface.

We also include a JavaScript library of common functions so authors can easily add calls to these JavaScript functions to simulate GUI interaction. For example, we provide functions for sorting bar charts and dynamic tables, selecting data within any of the views, and animating selection within bar charts. Readers may choose to record a sequence of interactions to be later replayed to create a customized version of the document.

### 3.3 An Extensible Framework

While the framework provides an efficient approach to authoring LiveDocs, it can also be easily extended to accommodate new views. More specifically, linking between existing views is handled through a form of publisher-subscriber methods. This means that any new view developed within the bounds of the linking model can then be added and linked to any existing views without requiring the modification or recompilation of any of the existing views. Thus, LiveDoc authors can use their own new views or combine our views with existing domain-specific views, providing more powerful analysis capabilities.

## 4. CORE LIVEDOC COMPONENTS

The current core set of LiveDoc components includes three basic types of views (bar chart, smoothed histogram, and dynamic tables). Each of these is described in more detail below.

*Bar Chart.* LiveDoc bar charts are used to indicate the number of cases (i.e., frequency distribution) for each value of a categorical variable. Linking a bar chart to other views provides added analytical power. Clicking on one or more of the bars allows users to select subsets of cases in all linked views. If another linked component, for example a table or another bar chart, is used to select a subset, each bar is partially highlighted according to the fraction of cases in that bar have been selected.

*Smoothed Histogram*‡. While LiveDoc bar charts are used to show the frequency distribution of categorical variables, we use the term "histogram" to describe a smoothed distribution of continuous variables. Selection within a histogram is very similar to the bar chart—users can select values via direct manipulation and selections made in other linked views are reflected through corresponding partial highlighting within the histogram.

*Dynamic Tables.* The LiveDoc dynamic table is modeled after the Table Lens [10] and provides a spreadsheet-like view of the data. Each column within a dynamic table contains a variable which is measured on each of the cases shown in the rows. The table allows panning and zooming, so that subsets of the cases can be hidden from view. Each column of numerical data is displayed using a collection of horizontal bars, one bar per cell, where the length of a bar is proportional to the numerical value of its cell. This allows the user to see trends across rows and relationships among columns. If the user has zoomed in far enough, the numeric values of the variables are also printed in the table (e.g., see Figure 1). Users can select subsets of cases via the mouse (after doing so, the selected rows appear in yellow, shown as light gray in this paper). By clicking on a column heading, the user selects the variable in that column.

*HTML Links and Controls.* Since we support the use of HTML and JavaScript for scripting interactions to the view(s), we enable authors to include any of the standard control widgets available through HTML: check box button, radio button, input box, or drop-down choice menu. In addition, users can attach a JavaScript function call directly to an HTML link. Examples of our LiveDoc JavaScript functions are described above in Section 3.2 and sample JavaScript links and control widgets are presented in our sample LiveDocs described in the next section.

*Implementation.* All LiveDoc components have been implemented as Java 1.0 applets. The components have not been ported to later versions of Java due to the constraint that many of our target LiveDoc readers are running web browsers that only support Java 1.0. The JavaScript controls access applets by invoking their public methods. This is referred to as LiveConnect in the Netscape browser and it works identically in another popular browser made by Microsoft.

## 5. REAL-LIFE SCENARIO

We illustrate our LiveDocs approach through a real-life scenario about the analysis of sports data on truck racing. The Craftsman Truck Series is one of the National Association of Stock Car Auto Racing's (NASCAR®) top racing series. The vehicles look roughly like commercially available pickup trucks but contain 700 horsepower engines and reach speeds in excess of 180 miles per hour on some tracks. The 1999 season consisted of 25 races, with approximately 35 drivers participating in each, and with a total of 120 drivers appearing in at least one race. Race data are interesting in that they arise from interactions between two groups: drivers and races. We obtained the data from NASCAR's web site, www.nascar.com. Information is also available through www.sears.com/craftsman.

‡ While bar charts are useful when variables take on a small number of values which might not have an obvious order, smoothed histograms are better for working with variables for which interesting subsets are generally continuous ranges. Potential interactions with these histograms include changing the amount of smoothness; they are constructed using an Epanechnikov kernel.

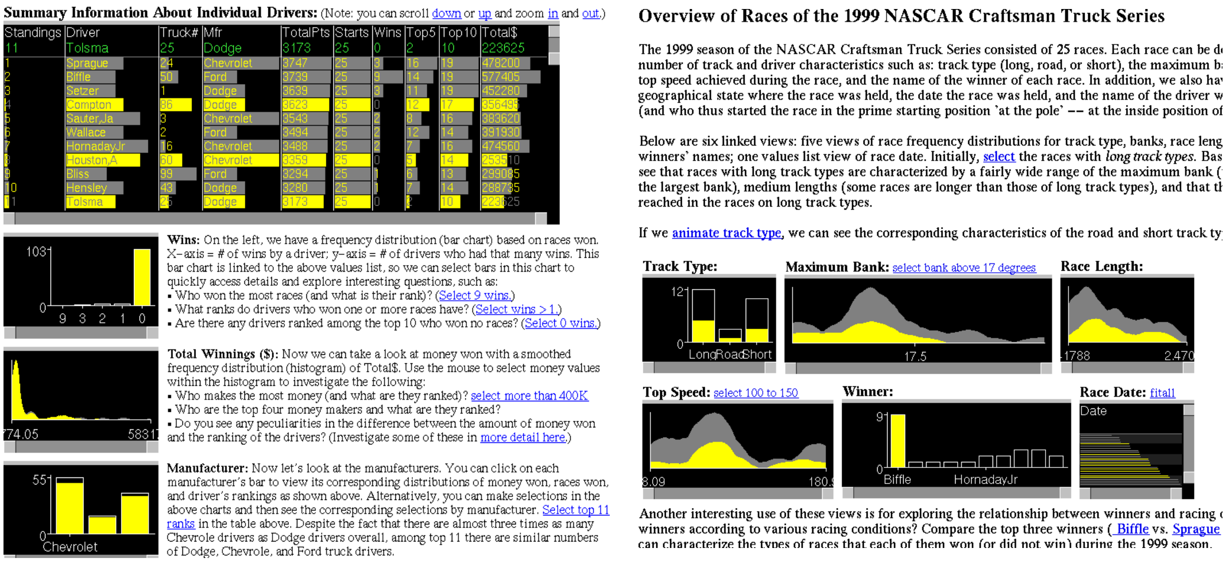


**Figure 1. Excerpts from two LiveDocs on the 1999 NASCAR® Truck Racing Season. The web page on the left summarizes information about individual drivers, while the web page on the right examines data characterizing the races.**

In the remainder of this section, we present some sample LiveDocs for analyzing the truck racing data described above. We present an excerpt from the static version of the LiveDoc exactly as it appears in the browser window (e.g., see Figures 1-2), discuss ways the user may interact with the infoVis components embedded within the LiveDoc and finally, describe how to compose such a LiveDoc.

## 5.1 Example 1: Overview of Drivers and Races

Figure 1 contains two LiveDocs, one which focuses on the drivers, and one which shows details about races. The first document includes a dynamic table containing information about the drivers such as their position in the season points standings, their truck manufacturer, and summary information about race results such as numbers of top ten finishes and total prize money. The dynamic table is linked to and followed by three additional views—a bar chart of drivers' number of wins, a smoothed histogram of drivers' prize money, and a bar chart of truck manufacturers.

The second document in Figure 1 focuses on the study of the truck races included in the 1999 season. Bar charts categorize the races by the track type (long or short oval or road course) and race winner. Smoothed histograms display distributions of angle of banking of the track, track length in miles and the speed of the track, as measured by the best time by drivers in qualifying. A dynamic table (in the lower right-hand corner of the six views) contains a single column listing information about race dates.

### *Potential User Interactions*

In the first document of Figure 1, the dynamic table initially displays only the top several drivers, but the text encourages the reader to zoom and scroll, e.g. to locate the two drivers who drove in only a few races but won one. The supporting views are useful for highlighting subsets of the data so that the reader can restrict attention to these subsets when viewing the dynamic table. For example, in the document shown on the left of Figure 1, the reader has selected drivers with zero wins in order to study how high in the standings it is possible to finish without the benefit of a win. The document also contains JavaScript controls (e.g., Select 0 wins) that the reader can click on to follow analyses recommended by the author. These controls are conveniently located within the text explaining their usage and in close proximity to the view itself.

Readers can also easily view the drivers with the most prize money by interacting with the "total winnings" histogram or the neighboring JavaScript control. This helps point out some surprising nonmonotonicities: e.g. the seventh place driver won almost as much as the champion by virtue of having won a \$100,000 bonus in the 100$^{th}$ Craftsman truck race. This overview page contains a link to a page with more detailed information in drivers' race results. Finally, the manufacturer bar chart can be used to compare manufacturers: for instance, relatively few drivers use Dodge trucks, but a high percentage of Dodge drivers are successful.

The second document of Figure 1 displays details about the races. This document demonstrates some ways in which JavaScript controls can enable exploratory analysis with minimal user interaction through animation: when the user clicks on animate track type, the control loops over the bars in the track type bar chart, selecting each in turn, and propagating the selection to the other linked views, so that the reader sees characteristics of each track type in turn. The text below recommends comparing the races won by the three drivers who won at least three races: while the three did not differ in preferred speed or track length, Greg Biffle seemed to prefer the flattest (least banked) tracks, Jack Sprague was best at driving at a severe angle, while Dennis Setzer had most moderate tastes.

### *Creating the LiveDoc Example*

In this section, we provide some technical details on creating the first sample LiveDoc of Figure 1. Excerpts of the HTML source for the LiveDoc web page are given below. It starts with HTML headers (containing definitions of relevant JavaScript functions), continues with the presentation text and includes controls in the form of links and views included via applet tags. While we provide general details describing how to author a LiveDoc in the Appendix, we explain some of the specifics of this example below. Due to space limitations, we do not include the full HTML source for the document, but rather highlight key examples and omit redundant text.

The first control, provided as a link in the text above the dynamic table, is a link that allows the reader to scroll <u>down</u> within the dynamic table:

```
<a href="javascript:doCommand('PAN VERTICAL -10','driverlist')">down</a>
```

The "javascript:" type tells that the link contains JavaScript code, "doCommand" is a utility function that invokes public method "doCommand" of an applet "driverlist". The first argument is the command to be executed by the applet.

The fifth control (i.e., the one for <u>Select 9 wins</u>) is also a link:

```
<a href="javascript:doCommand('REPLACESELECT ORDER SMALLEST','wins')
">Select 9 wins.</a>
```

It selects the bar on the left that contains drivers who won nine races.

The first applet tag describes the Dynamic Table view:

```
<applet name=driverlist code=spr.views.DTable.class  width=700 height=250>
<param name=url value="drivers.txt">
<param name="Variable"
value="Standings,Driver,Truck#,Mfr,TotalPts,Starts,Wins,Top5,Top10,Total$">
<param name="SortBy" value="Standings">
</applet>
```

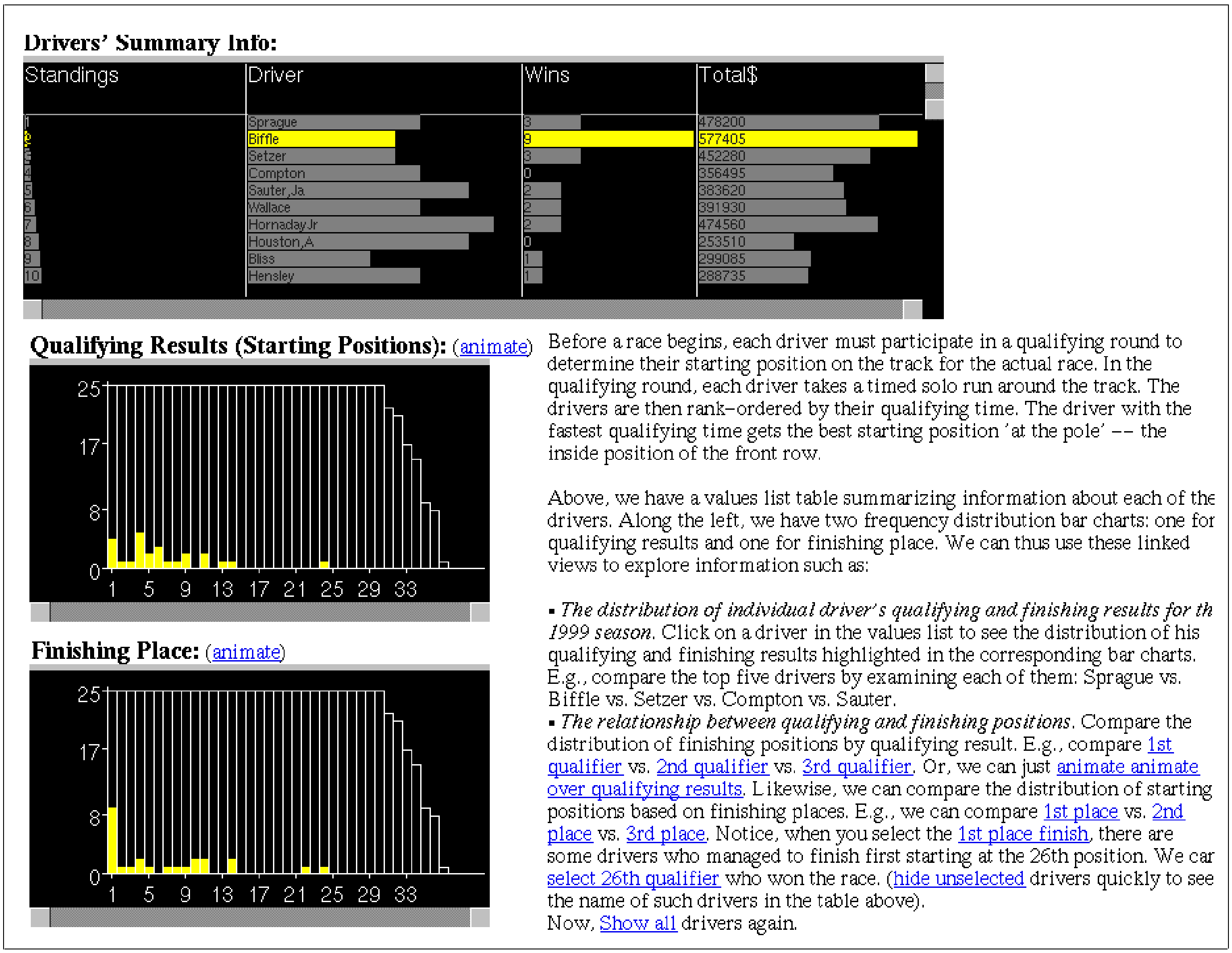


**Figure 2. A LiveDoc on detailed information from the 1999 NASCAR® Truck Racing data.**

The “url” parameter specifies a URL for the data to be displayed in the table. The “Variable” parameter lists the variables to be displayed in the table. The last parameter indicates that the table will initially be sorted by the column named “Standings”.

The second applet is a bar chart of number of wins:

```
<applet name=wins code=ldoc.BarApplet.class width=175 height=130>
<param name=url value="drivers.txt">
<param name="Variable" value="Wins">
</applet>
```

### 5.2 EXAMPLE 2: Relationship between Qualifying and Final Results

The next LiveDoc was designed to investigate the drivers’ results and qualifying performance in greater detail. The top table contains brief summary information about the drivers and can be used to select subsets of drivers to highlight the two bar charts below. The two bar charts contain finishing position and qualifying position information (qualifying consists of a single solo lap around the track by each driver and determines the order in which drivers start the race). By selecting a single driver (e.g., Biffle, as shown in Figure 2) one sees distributions of his qualifying and finishing positions. By selecting the top several drivers on the top list, one can investigate the hypothesis that top drivers tend to qualify better than they finish, since more uncontrollable events occur in the course of an entire race.

Also, one can study the effect of starting position on finishing position. There is naturally a trend in which faster qualifiers tend to finish better, but a surprising result appears when the user clicks the JavaScript control to animate the qualifying results. The distributions of finishing position for drivers that start in given positions appear to oscillate back and forth, with odd numbered starting positions being more favorable. Odd and even positions differ because the race begins with the drivers in two files, with the odd numbered qualifiers on the inside of the track. Furthermore, if we select the winners, we see that one driver managed to win, even when starting at the 26$^{th}$ position (in the qualifying results chart). By selecting the 26$^{th}$ position in “intersect” mode (using shift-click, or “INTERSECTSELECT” command), we can immediately identify the driver who managed to accomplish that task.

## 6. LIVEDOC USAGE AND OTHER APPLICATIONS

The LiveDoc framework has evolved over the past couple of years, primarily through its use in a project focusing on understanding and tracking the development of large software systems. Over 100 LiveDocs were created as part of this project (in this paper we present only examples with nonconfidential sports data). These LiveDocs were used to facilitate collaboration among researchers on the project, as well as to disseminate results to researchers inside and outside of the project. In addition, LiveDocs containing summary information of key results were presented to middle- and upper- management of the software development departments being studied. The results provided managers with feedback on how the software of their department had evolved over time as well as hints to how they might restructure their code or their organizations to improve the software engineering process.

The early version of our LiveDocs framework did not allow authors to set the initial state of LiveDoc view components. When readers reviewed these older LiveDoc web pages, they could read about key results, but they were presented with visual results that did not initially highlight or match the textual description of results. This was particularly problematic for managers, who often did not have time to interactively select or explore the data on their own. This user feedback led to the additional applet parameter for setting a view’s initial state. The ability to set the initial state of the view applets allows us to present key results upfront while still preserving the ability to let users explore the data on their own (and in context), if they are inclined to do so.

Earlier versions of our LiveDocs framework also included control widgets as separate Java applets rather than as JavaScript components. While the concept and functionality of the Java applet and JavaScript control widgets are very similar, the use of JavaScript has the added advantages of smaller size and greater extensibility. That is, with the use of JavaScript, readers do not have to wait while extra applets are being loaded, and authors do not have to program new controls in Java when a new view or functionality is added. The move to JavaScript necessitated the introduction of a simple language to script the views (since JavaSript can pass strings to Java methods, but cannot directly create Java objects). Such a language can be used to script arbitrary user interactions and to provide an alternative to a GUI interface.

The use of LiveDocs within the above project on analyzing large software systems also led to the addition of new LiveDoc views that were tailored to particular problem domains. They included geographic and abstract layouts and views to display software code and changes. Most of the applications, however, did not require construction of additional domain specific views.

In addition to the LiveDocs for presenting the analysis of truck racing data (Section 5) and the LiveDocs for characterizing large software systems described above, we have also applied the LiveDocs approach to the analysis of organizational data and to a case study of the Ty Company’s success with the Beanie Baby collectible toys.

In the future, we plan to provide a new, direct-manipulation interface for authoring LiveDocs. This will be done within the context of a larger project referred to as InfoStill (short for information distillery; see [14] for more details). In the InfoStill framework, authors will have a GUI interface for creating LiveDocs and the HTML, including applet tags, will be automatically generated for them.

## 7. RELATED WORK

Commercial or free Java applets of simple charts such as bar charts, pie charts, etc. are available today (e.g., [7]). Such charts have the advantage of dynamically displaying the latest version of data, allowing authors to include simple charts without having to program them by hand, while appearing like a typical static document and thus easily accessible to readers. Although some of these charts support some user interaction, such interaction is typically very limited (e.g., panning in 2-D charts, rotating of 3-D bar charts). Also, to our knowledge, none of these existing applets support linking between views and thus lack the analytical capability that is available through LiveDocs. That is, the power of the LiveDocs approach is that it provides both access to simple views and support for conducting sophisticated analysis through data exploration within a linked views paradigm.

As mentioned in the introduction, more sophisticated web-based infoVis applets are becoming available for visualizing and exploring data such as financial or geographical data [11], [9]. Unfortunately, these more complex visualizations tend to be tailored to a particular domain and/or are larger in size. Thus, while such views may provide unique analysis capabilities, they do so at the expense of requiring users to learn a new interface.

Much of the research conducted within the infoVis community has focused on creating new visualizations or interfaces to visualizations to support users in accessing or analyzing data via direct manipulation (e.g., [1], [3]). While some attention has been given to the problem of automating the presentation or construction of appropriate visualizations (e.g., [8]), such work has not taken the notion of visualization in context (e.g., with textual descriptions or annotations) into consideration. Previous work on analyzing the types of tasks users conduct during data analysis through infoVis indicates that users consider presentation-related tasks (i.e., creating and describing a presentation of results) to be both important and time-consuming [6]. Our LiveDocs approach is designed to support authors in easily creating and customizing such presentations of results in context, while still providing interactivity to enable readers to easily investigate claims made by the authors.

## 8. CONCLUSION

In this paper, we presented our current LiveDocs framework for providing a simple yet powerful information visualization platform targeted to users that have limited to moderate motivation to use sophisticated visualization systems.

The framework is based on a set of visualization components that are used to compose domain specific Web pages via simple HTML authoring. The parameters of the components allow authors to specify linking among the components, and to initialize and customize the set of controls for the component.

We illustrated our framework through actual LiveDoc examples applied to real-life sports data. Our examples illustrate how setting the initial state of LiveDoc components provides visual support in presenting key data analysis results; linked interactive views allow readers to further confirm and explore results on their own; and author-scripted interactions presented in context, engage the reader and minimize the learning effort.

## 9. ACKNOWLEDGMENTS

This research was supported in part by National Science Foundation grants SBR-9529926 and DMS-9208758 to the National Institute of Statistical Sciences.

## 10. REFERENCES

[1] Ahlberg, C. and B. Shneiderman. (1994). “Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays.” In *CHI’94 Conference Proceedings*. New York, NY: ACM Press, 313-317.

[2] Appelt, W., Hinrichs, E. and G. Woetzel. (1998). “Effectiveness and efficiency: the need for tailorable user interfaces on the Web.” In *WWW’98 Conference Proceedings*.

[3] Carlis, J.V. and Konstan, J.A. (1998). “Interactive Visualization of Serial Periodic Data.” In *UIST’98 Conference Proceedings*. New York, NY: ACM Press, 29-38.

[4] Eick, S.G., Mockus, A., Graves, T.L. and A.F. Karr. (1998). A Web Laboratory for Software Data Analysis. In *World Wide Web, 1(2),* 55-60.

[5] Fischer, G. and Girgensohn, A. (1990). “End-user modifiability in design environments.” In *CHI’90 Conference Proceedings*. New York, NY: ACM Press, 183-192.

[6] Hibino, S. (1999). “Task Analysis for Information Visualization.” In *Third International Conference on Visual Information Systems (VISual’99).* (Huijsmans, D.P. and A.W.M. Smeulders, Eds.). Berlin: Springer-Verlag, 139-146.

[7] KL Group Inc. (1999). JClass Java Components. http://www.KLGroup.com/

[8] Kolojejchick, J., Roth, S.F., Lucas, P. (1997). “Information Appliances and Tools in Visage.” *IEEE Computer Graphics and Applications*, July/August, 31-41

[9] Professional Geo Systems. (1999). LAVA GIS Browser. http://www.pgs.nl/

[10] Rao, R. and S.K. Card. (1994). “Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus Plus Context Visualization for Tabular Information.” In *CHI’94 Conference Proceedings*. New York, NY: ACM Press, 318-322.

[11] SmartMoney.com (1999). SmartMoney’s Map of the Market. http://www.smartmoney.com/marketmap

[12] Tufte, E. R. (1983). *The Visual Display of Quantitative Information*. Cheshire, CT: Graphics Press.

[13] Wills, G.J. (1999). “Natural Selection: Interactive Subset Creation.” *Journal of Computational and Graphical Statistics*. To appear.

[14] Cox, K., Hibino, S., Hong, L., Mockus, A., and Wills, G. (1999). “InfoStill: A Task Oriented Framework for Analyzing Data through Information Visualization”. *Proc. IEEE Information Visualization Symposium Late Breaking Hot Topics*, pages 19-22, October 1999.

## APPENDIX: AUTHORING LIVEDOCS

The technology for composing a live document is in many ways similar to the technology for creating a regular presentation or report. A favorite word processor or HTML editor may be used to create the text and format the presentation. The illustrations (at this point, dummy images corresponding to the initial state of our interactive views) can be placed in the appropriate places within the document. Once that is completed, the document can be exported to HTML format, if necessary, and the dummy illustrations and controls can be replaced by the actual views. A significant departure from creating a regular document is the necessity to decide which interactions with which views to provide to readers and where to place the appropriate controls.

Our base prototype collection includes Bar Chart, Histogram, and Dynamic Table views. When adding a view to a LiveDoc, authors must specify the type of view and data source parameters and may optionally set an initial state of the view and information for linking views together. The table below describes the different types of view configurations. The parameters for the ***applet*** tag are specified in name/value pairs. Both the name and the value are strings of text.

If the views share the same string in the ***url*** parameter, those views are linked together in the sense that when the user highlights a subset in one view, it is automatically highlighted in another view. Also, views that have a common ***url*** parameter do share the same data, thereby reducing the download time of the presentation. The `Join` parameter allows linking of data from two data sources by specifying variables to match in the two sources.

The command language represents a string based alternative to user GUI interactions. It includes commands for panning and zooming, selection of subsets, sorting columns or bars, and finding a subset of records with specified values.

## Table A1. View Parameters.

| **Parameter** | **Effect** |
|---|---|
| *Data Source* | |
| ***url*** | set the data source |
| ***Show*** | select a subset of data records to view |
| ***Variable*** | identify which data fields to show |
| *Initial State* | |
| ***SortBy*** | set sort criteria for the initial state |
| ***Highlight*** | highlight a subset of the data |
| ***Transform*** | control visual representation of the data |
| ***Show Policy*** | display all case or only selected cases |
| ***doCommand*** | execute a command that changes the state of the view |
| *Linking Views* | |
| ***Join*** | link views from different data sources |