Working with the Echo Web framework, Part 1: An introduction to Echo

Learn the basics of building Web-based applications that work like rich clients

Level: Introductory

Tod Liebeck (tliebeck@nextapp.com)
Chief Software Architect, NextApp, Inc.
September 9, 2003

This two-part series provides an introduction to the Echo framework, an open source, Java technology-based platform for building Web applications that look and act like rich clients. Part 1 introduces the framework and discusses what it does and how it is best used, providing an introductory walkthrough of its features. Part 2 takes you more in depth, building on your knowledge from Part 1 to develop a complete application using the Echo framework.

Of rich clients and Web clients

Web-based applications are typically built as collections of stateless services. The client -- a Web browser -- invokes a service by making an HTTP request. The server sends back a dynamically generated HTML page that is then displayed in the browser. The generated document reflects the state of the application, providing links and forms that enable the user to invoke other services.

When requirements call for greater user interface capability, such Web-based applications often come up a bit short. In these cases, it's common to see projects move to rich-client architectures; the low deployment costs of a Web-based architecture are sacrificed to get out from under the limitations of the Web. The problem is not a lack of capability on the part of the browser, but rather the fact that its tools are difficult to use to create complex applications. As many developers have learned the hard way, HTML and JavaScript are not particularly modular or maintainable in real-world development.

In a nutshell, the Echo framework provides a Web-based platform that offers rich-client levels of capability and maintainability. Echo developers create applications using a component- and event-based API that resembles a rich-client user interface toolkit (think Java Swing). A translation layer generates the HTML required to render the user interface to the client browser. To bring the experience closer to the standard of a thicker client, Echo uses an extensive library of client-side JavaScript that extends the capabilities of the browser. The architecture aims to completely remove the end developer from having to think about the management of multiple

windows, frames, forms, and complex DHTML and JavaScript used to create a sophisticated Web application.

Under the hood

Echo provides a high level of abstraction between the development of a Web-based user interface and the requirements of the browser environment. To achieve this abstraction, the framework is built in two distinct tiers. The first tier, called the *Component Framework*, provides the developer with a user interface toolkit API with which applications are built. The second tier, the *Application Container*, automatically translates the state of an application into the languages of the browser.

When working with the Component Framework API, a developer is completely unconcerned with the usual concepts involved in developing Web-based applications, such as parsing HTTP requests and rendering HTML and JavaScript code. The Component Framework tier of Echo is entirely uninvolved in any client-related aspects of the Web application. All such responsibility is taken on by the Application Container.

Develop an application with Echo
Part 2 of this article walks through the development of a complete application built using the Echo framework -- a Web-based e-mail client.

The Application Container dynamically creates peer objects for each user interface component that is presently a part of the application. These peers function as intermediaries between the user interface components and the client Web browser. Each one translates the state of its represented component into HTML and JavaScript as required, and fires events to the component in response to information received in HTTP requests.

Web browser interaction

On the client side, Echo differs radically from the usual architecture of a Web application framework. Client-side JavaScript is used to ensure that the state of the client is synchronized with that of the server. Each time the user takes an action that requires the Web client to contact the server, all of the changes the user made are sent in a single HTTP request, even if they were made in different forms in multiple frames or windows. Other information is sent as well, such as the positions of scroll bars and checksum-like data used to ensure that the client state matches the server-side representation.

To make this work, Echo adds a non-visible HTML frame to each of an application's windows on the client Web browser. This "controller" frame is used as a communications channel between the client and the server. All actions taken by the user between server interactions are recorded in hidden form elements in the controller frame. When the user makes a request that will require server interaction -- such as clicking a button that has `ActionListener`s -- the controller frame submits all the changes to the server at once.

On the server side, the Echo Application Container processes this incoming HTTP request, firing events to components that were updated by the user. The Application Container tracks which components have been visually changed as a result. Finally, a new controller frame document is created that contains JavaScript directives to instruct the client to re-render the updated content.

This design removes many of the unusual constraints and annoyances that often show up in Web application development. The problems that would normally occur when user-input fields are spread across multiple windows and documents are automatically eliminated. Updating content in multiple frames and windows at the same time no longer requires any extra work. Scroll bars don't jump back to the top every time a frame is updated. The list goes on, but the overall effect is that both the developer and the user enjoy a more rich client-like application experience.

What you'll need

To get started with Echo, you might need to obtain a few new tools. In addition to Echo itself and JDK version 1.3 or higher, you will need:

- **Apache Ant.** Ant is a Java technology-based build tool used to compile and package applications. In our case, we'll use it to simplify the task of compiling a few tutorial applications and creating deployable Web archives (WAR files).

- **A servlet container.** Echo applications are servlets, and therefore require a servlet container to make them work. Any container that meets the servlet 2.2 specification should work. If you're new to the servlet world, try out a recent version of Jakarta Tomcat from the Apache group (see Resources). It's free and runs on most platforms.

The elements of an Echo application

In many ways, creating an Echo application is similar to creating a thick-client using Java Swing. The state of an application's user interface is represented as a hierarchy of Component objects. Event listeners are added to these components that perform work in response to a user's actions.

It is the responsibility of the Application Container to ensure that the end user of an Echo application will see an up-to-date rendering of the application's component hierarchy at all times. The Application Container uses `PropertyChangeListener`s to track visual changes made to any components visible within the application. Each time the client makes a request to the server -- causing the component hierarchy to be updated -- the state will be subsequently reflected back to the client. The underlying lesson here is that all you have to do to make a change in what a user will see is to make that change to the component hierarchy. The Application Container takes care of the dirty work.

The heart of an application: The EchoInstance object

The state of a single user's instance of an Echo application is represented by an `EchoInstance` object. When a new user visits an Echo application, a new `EchoInstance` is automatically created by the Application Container and stored in the user's servlet session. An `EchoInstance` is thus a stateful object, and as such is often used to store application-wide state information. For example, the name of the currently logged-in user could be stored in an instance variable of the `EchoInstance`, or perhaps it might also be used to store references to open transactions or other middle-tier and database resources.

`EchoInstance` functions as the effective "root" of the user interface. It is responsible for tracking and managing which windows are open in the application, providing a complete picture of the application's state.

Every Echo application is required to provide an implementation of the abstract `EchoInstance` class. The derived form is only required to provide an implementation of a single method, `init()`, which initializes the state of a new application instance. The `init()` method returns a `Window` object that represents the state of the "initial window" of the application. The component hierarchy placed in the "initial window" is rendered in the browser window used to first navigate to the application.

A very simple implementation of an `EchoInstance` is shown in Listing 1, displaying the text "Hello, World!" in the user's browser window:

Listing 1. A simple EchoInstance implementation

```
class HelloWorld extends EchoInstance {

    public Window init() {
        Window window = new Window();
        ContentPane content = new ContentPane();
        window.setContent(content);
        Label label = new Label("Hello, World!");
        content.add(label);
        return window;
    }
}
```

The EchoServer: A servlet wrapper for an Echo application

To make Echo applications work as servlets, a servlet class must be defined to house the application. Echo's `EchoServer` class (which extends `HttpServlet`) handles nearly all of this work right out of

the box. For instance, `EchoServer` provides implementations of the `doGet()` and `doPost()` methods to handle incoming requests automatically by feeding them to the Echo Application Container.

The only task required of the developer when creating an `EchoServer` implementation is to tell the object how to create new user instances of an application. This is accomplished by providing a concrete version of the abstract `getEchoInstance()` method to return a new application instance on demand. Listing 2 shows just how simple such a class can be:

Listing 2. An EchoServer implementation

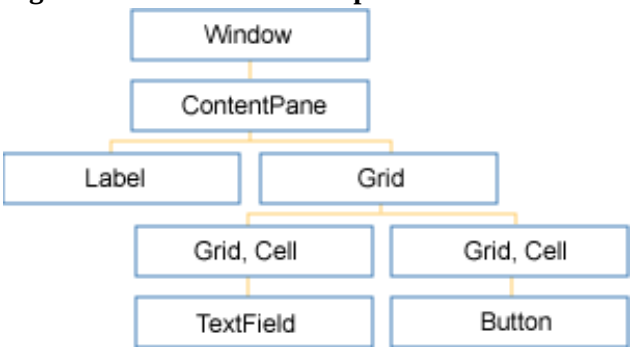
```
public class HelloWorldServlet extends EchoServer {

    public EchoInstance newInstance() {
        return new HelloWorld();
    }
}
```

Building a user interface with components

An Echo `Component` is an object that represents an element or "widget" of a user interface. For example, a label component is used to display a string of text or an icon, and a text field component is used to enable the user to input a single line of text. Components may contain child components, enabling the formation of component hierarchies. An Echo user interface is defined by such a component hierarchy, as shown in Figure 1:

Figure 1. A user interface represented as a hierarchy of components



Echo provides a basic array of built-in components to cover the fundamental concepts of user interfaces, such as windows, labels, buttons, text entry areas, listboxes, and tables. Components are designed to be extended so that developers can derive new reusable or application-specific components from the existing library.

Responding to user actions through events

While the state of a user interface is defined by components, its dynamics are built using events. Components that are interactive -- for instance, buttons -- provide methods to register event listeners. When an event listener is registered with a component, it is notified when the component changes state in response to a user's behavior. For example, `ActionListener`s registered with a `Button` component will be invoked when the user clicks the button.

As previously mentioned, Echo is an event-driven framework. Once the initial state of an application has been set up as a result of `EchoInstance`'s `init()` method being invoked, all future changes to an application's state will be in response to events.

Listing 3 shows the `EchoInstance` class of a very simple application that demonstrates the use of events. It displays three colored buttons that change the background color of the window's content when they are clicked. The class itself implements the `ActionListener` event-listener interface, providing an `actionPerformed()` method to process action events. When the class creates the buttons, it adds

itself as an action listener to each of them.

Listing 3. The EchoInstance of an application that makes use of events

```
class ButtonDemo extends EchoInstance
implements ActionListener {

    private Button redButton, greenButton, blueButton;
    private ContentPane content;

    public void actionPerformed(ActionEvent e) {

        if (e.getSource() == redButton) {
            content.setBackground(Color.RED);
        } else if (e.getSource() == greenButton) {
            content.setBackground(Color.GREEN);
        } else if (e.getSource() == blueButton) {
            content.setBackground(Color.BLUE);
        }
    }

    public Window init() {

        Window window = new Window();
        content = new ContentPane();
        window.setContent(content);

        redButton = new Button("Red");
        redButton.addActionListener(this);
        redButton.setBackground(Color.RED);
        content.add(redButton);

        greenButton = new Button("Green");
        greenButton.addActionListener(this);
        greenButton.setBackground(Color.GREEN);
        content.add(greenButton);

        blueButton = new Button("Blue");
        blueButton.addActionListener(this);
        blueButton.setBackground(Color.BLUE);
        content.add(blueButton);

        return window;
    }
}
```

When a button is clicked, the `actionPerformed()` method is invoked. The implementation of this method determines which button was pressed by checking the source of the event and then setting the background color of the application based on which button fired it.

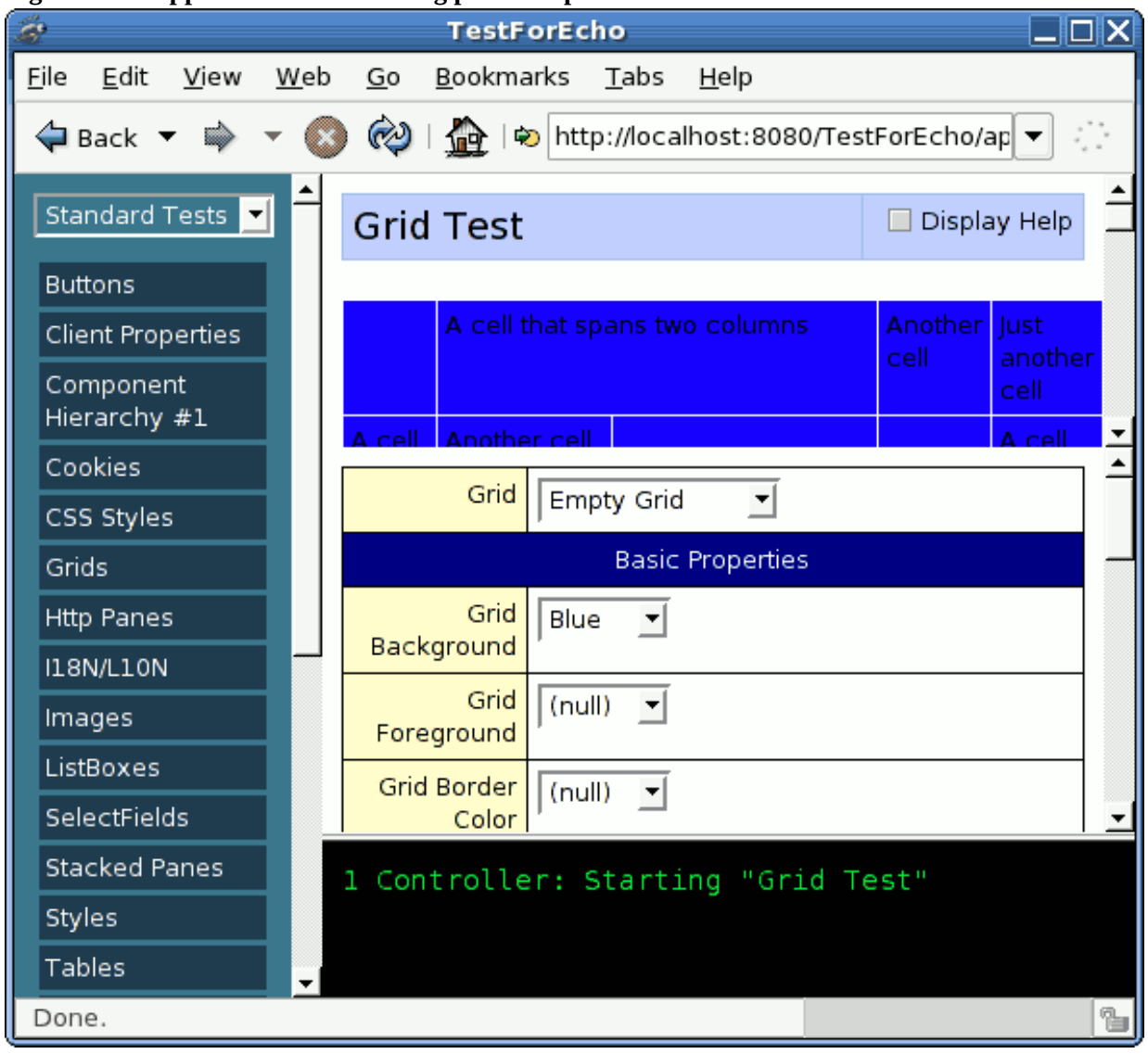
Laying out the user interface

By default, Echo components will be laid out sequentially, from left to right and then top to bottom on the user's screen. This default layout is not desirable when building a real application. Echo provides numerous components whose purpose is to lay out the user interface of an application, such as defining the positioning and order of other components.

The "strut" component is arguably the simplest of Echo's layout components. Its sole purpose is to provide horizontal or vertical spacing between two other components. Struts are created by invoking a static factory method in the `Filler` class. Two such methods exist: `createHorizontalStrut()` and `createVerticalStrut()`. Each takes an integer value as a parameter to indicate the size -- in pixels -- of separation desired.

Most application layouts are typically defined using Echo's `Grid` class. `Grid`s provide a greater degree of control by letting the developer place components in the cells of a grid. Figure 2 shows an application using several `Grid` components to lay out its content. The buttons on the left side of this figure are laid out using a `Grid`, as is the content in the central region of the application's window.

Figure 2. An application laid out using pane components and Grids



Windows and panes

In Echo, the `Window` component represents a complete and independent browser window. In contrast to most Web-based application architectures, Echo allows the developer complete control over management of multiple windows involved in a single application. `Window`s may be opened and closed by invoking the `EchoInstance`'s `addWindow()` and `removeWindow()` methods. Applications can even be notified when the user attempts to close a window by registering `WindowListener`s.

Pane components are used for dividing `Window` components into smaller regions. `Window`s are

somewhat unique from other components in that they only allow a single component as their content, and that component must be a pane. All pane components are recognizable by the fact that they are derived from the `AbstractPane` class, in addition to having the "Pane" suffix in their class names.

The two most commonly used pane types are `ContentPane`s and `ContainerPane`s. A `ContentPane` allows regular components (that are not panes) to be added, such as `Label`s, `Panel`s, `Grid`s, and `TextField`s. In both the HelloWorld (see Listing 1) and ButtonDemo (see Listing 3) examples, `ContentPane`s have been used as the root content of each application's main window.

`ContainerPane`s are used to divide a window's "real estate" into distinct areas. `ContainerPane`s may only contain other pane components (including other `ContainerPane`s). In Figure 2, the window has been divided into major regions using several `ContainerPane`s.

Style properties

Most Echo components provide settable properties that control their appearance, as shown in Listing 3, where the background colors of the `Button` and `ContentPane` components are modified. At a minimum, every component is capable of adjusting its font, as well as its foreground and background colors, because the `Component` class itself provides these features. Many components contain additional stylistic properties. For example, `TextField`s allow you to configure the style, size, and color of their borders.

When developing a complex user interface, you will often want the same properties applied to many instances of the same type of component. For example, you might want all text fields to have a light blue background, black text, and a thin dark blue border. Echo provides a `Style` class to group such property settings, so that the same information can be reused by multiple components.

Applying Echo

As you've learned in this article, Echo has a lot in common with a user interface toolkit. As a result, Echo works best with applications whose user interfaces lend themselves to being built with a user interface toolkit-like architecture. If a project works well as a rich client application, it will likely work well with Echo; but if you think a project resembles a Web site more than an application, you will probably want to use a different technology to build it.

Many projects will benefit from using a hybrid of Echo and other technologies. It's fairly common among the Echo development community to see projects built using both Echo and other Web application frameworks and Web scripting languages. These sites are often architected so the page-based technologies are used to create the content-rich portions of the site, and Echo-based applications are created to handle the more complex and interactive pieces.

The future of Echo

After more than 20 months of development and testing, Echo 1.0 was finally released in June 2003. There are two new versions of Echo currently in development: 1.1 will offer some minor updates and is set to be released in the near future; 1.2 will offer more significant improvements and will be released a few months later.

For 1.2, a roster of new features is planned. Many of the additions are aimed toward providing a more rich client-like experience, such as allowing for complete control over component focus, as well as tracking and updating window screen positions and sizes. Other features target making life easier on developers, such as allowing style information to be externalized and streamlining the installation process of third-party Echo component libraries.

Because Echo is an open source project, all development takes place out in the open. The primary vehicles of communication among developers are the online forums and mailing lists. Anyone is welcome to join and contribute or discuss ideas for future development.

Resources

- The Echo home page provides useful resources for Echo developers.

- Download the Echo Web application framework. This page provides links to download the latest versions of the Echo libraries. You'll need them to create Echo applications of your own.
- The Echo online tutorial provides an in-depth overview of the fundamentals of Echo application development.
- The Echo High-Level Technical Overview provides a detailed explanation of how the inner architecture of Echo works.
- The Apache Ant home page provides information about Ant and the latest versions for download. Ant is used to build the sample applications used in this article.
- Just getting started with Ant? In this excerpt from *Java Tools for Extreme Programming* (Wiley, 2002), Rick Hightower and Nick Liesecki offer guidance on how to build Java applications with Ant.
- The Apache Tomcat home page provides information about the Tomcat servlet container and the latest versions for download.
- Find hundreds of Java technology-related resources in the *developerWorks* Java technology zone.

About the author
Tod Liebeck founded NextApp, Inc. in 2001 to pursue advanced Web-based user interface technology. He currently serves as the lead developer of the Echo Web application framework and Chief Software Architect. Contact Tod at tliebeck@nextapp.com.