Cross-browser Internet applications with OpenLaszlo

IN A FLASH

Long before Ajax hit the scene, Flash gave developers a basis for designing interactive web content. The OpenLaszlo framework, with its easy-to-learn command syntax, is the foundation on which many Flash applications build. **BY PETER KREUSSEL**

Flash does not have many friends among the open standard lobby on the Internet. This said, if you have no objection to using the closed-source Flashplayer, the OpenLaszlo framework offers a powerful, easy-to-use platform for interactive Internet applications with sophisticated graphics. OpenLaszlo creates Flash bytecode based on an XML language that uses HTML-style tags for interface design and plain Javascript for the client-side application logic. Flashplayer 7 for Linux (which has been around for a while) is all you need to run the applications without restrictions.

The early preview of OpenLaszlo version 4 can convert applications to Ajax code, and thus do without proprietary technology. However, this does not work for the full feature set, and thus far, only Firefox/Mozilla and Internet Explorer 6 have been able to render OpenLaszlo DHMTL applications reliably.

Adobe's Flash has a number of technical benefits compared with Ajax. Adobe [1] claims that more than 95 percent of all browsers have a Flash plugin installed, and you can assume that just as many surfers disable Javascript as have browsers that do not support Flash. Flash technology, which supported interactive applications before Ajax hit the scene, offers a number of advantages:

- Flash avoids browser compatibility issues. Flashplayer will render a .swx file identically in Firefox, Opera, Internet Explorer, or other browser that supports Mozilla or Internet Explorer plugins. Even if Ajax applications are based on libraries or frameworks that introduce an abstraction layer between content and browser capabilities, achieving comparable robustness means much cross-browser testing.
- Flash applications can integrate many media types, such as animations, vector graphics, and sounds.

OpenLaszlo applications are based on XML files. The OpenLaszlo language, LZX, defines the user-interface design and provides a basis for client-side application logic, which keeps barriers low for newcomers by integrating both HTML and CSS elements. The language is XML compliant and object oriented, which means that you can reuse code

Listing 1: Object Orientation

```
01 <class name="box" height="100"
02 width="100" bgcolor="red"/>
03
04 <class name="borderedbox"
   extends="box">
05  <view bgcolor="yellow"
06     x="3" y="3"
07     width="${parent.width-6}"
08     height="${parent.
   height-6}"/>
09 </class>
10
11   <box/>
12   <borderedbox/>
13   <borderedbox/>
```

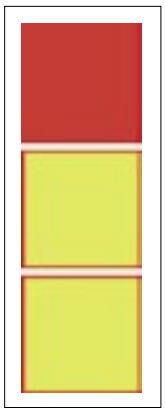

Figure 1: Object-oriented interface designs.

effectively for your interface design. Listing 1 creates the display shown in Figure 1.

The listing starts with the definition of the *box* and *borderedbox* classes, which can be displayed as often as you like by calling <*classname*> in the source code. *borderedbox* inherits from the *box* class; at the same time, the appearance of *borderedbox* is redefined on the basis of the inherited properties.

GUI Design

Let's look at a couple of examples to demonstrate how powerful the XML interface design language is. <*datepicker*> </*datepicker*> is all it takes to create a sophisticated date picker. Menus that work reliably, no matter what browser you choose, are not much more complicated. Listing 2 shows the XML code for a drop-down menu.

A multicolumn list (grid), which sorts the entries in a column when you click on the column header and supports editing or selecting in rows, is just a couple of lines of code. See Listing 3. See the OpenLaszlo homepage [2] for an overview of the available form elements.

Animations make life easier for users, especially if a developer needs to visualize a state change from state 1 to state 2. OpenLaszlo makes effects simple – to create a smooth movement, all you need do is call the *animate* method in any display element. In Listing 4, the *onclick* handler for the *box* element takes care of this, causing a *view*-type element – a square "window" with an original height of 20 pixels – to grow to a height of 200 pixels in 500 milliseconds. Clicking a second time shrinks the box again. The instructions for this are contained in the parameter *box height = =20?200:20*, and the *500* parameter defines the duration of the animation.

OpenLaszlo uses Javascript for application logic. Besides global <*script*> blocks, the framework supports methods that provide specific functionality for individual display elements or element classes. Event handlers call Javascript code while keeping to Javascript standards.

The following example, which moves a button five pixels to the right when clicked, provides an idea of how OpenLaszlo handles user input. The *button* tag defines an *onclick* handler, which calls the *moveHoriz()* function with an argument of *5*. After adding the label *Move me*, the sample code implements the *moveHoriz* method, which increments the *x* position by the number passed in as a parameter, again using syntax borrowed from Javascript. The code keeps to existing Internet standards, simplifying life for newcomers:

```
<button onclick=➥
"moveHoriz(5);">
  Move me
  <method name="moveHoriz"➥
args="moveAmount">
     this.setAttribute("x",➥
this.x+moveAmount);
  </method>
</button>
```

The constraints in *${DOM path to input field}* offer another option for responding to user input:

```
<checkbox id="cbox"ßß
  text="Show Window"➥
   x="10" y="10" />
<window visible=ßß
"${cbox.value}" />
```

The value of the window attribute, *visible*, is bound to the current value of the *cbox* checkbox by a constraint. When a user enables or disables the checkbox, OpenLaszlo will modify the visibility of the window accordingly, without needing an event handler to do so.

Data Handling

The core task of many web applications is that of displaying or editing data saved server-side. To support this, the OpenLaszlo XML language integrates a number of powerful methods.

The grid display element draws the displayed elements from the superordinate dataset element. The *dataset* tag contains either the XML-formatted data, as in Listing 5, or the *dataset* element loads the data from the server via HTTP, as shown in Listing 3. This gives developers the ability to evaluate current database data.

OpenLaszlo processes data on the basis of XPath [3], a technology developed by the W3C consortium to address parts of an XML document. In the example, */forecast/day* in *contentdatapath*, which is in the data source, addresses all *day* nodes in the *forecast* category.

Client-Server Interaction

Interactions between client- and server-side program logic are typical for larger scale web applications. In many cases, access by multiple clients to a shared database requires access to functionality implemented server-side, often with the aim of keeping the client application lean. A lean client application saves

Listing 2: Drop-Down Menu

```
01 <menubar width="200" >
02   <menu text="Menu 1" width="100">
03    <menuitem text="Menu item 1"
04 onselect="canvas.whichOne(this);"/>
05    <menuitem text="Menu item 2"
06 onselect="canvas.whichOne(this);"/>
07    <menuitem text="Menu item 3"
08 onselect="canvas.whichOne(this);"/>
09    <menuseparator/>
10    <menuitem text="Menu item 4"
11 onselect="canvas.whichOne(this);"/>
12   </menu>
13   <menu text="Menu 2" width="100">
14    <menuitem text="More items..."
15 onselect="canvas.whichOne(this);"/>
16   </menu>
17 </menubar>
```

Listing 3: Multicolumn Grid

```
01 <canvas height="250">
02  <dataset name="weatherdata"
   request="true"
03    src="http://www.
   laszlosystems.com/
04 cgi-pub/weather.
   cgi?zip=10022"/>
05  <grid datapath=
   "weatherdata:/weather"
06 contentdatapath=
   "forecast/day"/>
07 </canvas>
```

Listing 4: State Change

```
01 <canvas>
02   <view id="box" width="200"
   height="20"
03   <bgcolor="red"/>
04     <text onclick="box.
   animate(
05 'height',box.height==
06    20?200:20, 500, false)">
07       Click here
08     </text>
09 </canvas>
```

client resources and saves downloading large amounts of program code when launching an Internet application.

Besides data access, OpenLaszlo offers various approaches to integrating server-side processes with the client application flow. One way of establishing a connection to a server in the background is the *XMLHttpRequest* class, which OpenLaszlo provides in the form of *script* blocks and Javascript implementations for current browsers.

In contrast to Javascript code, which runs in the browser, the similar OpenLaszlo code does not pose any issues with browser incompatibility. OpenLaszlo converts the Javascript code to Flash bytecode that the Flashplayer, rather than the browser's Javascript engine, runs.

Besides *XMLHttpRequest* in Javascript code, OpenLaszlo also supports the SOAP protocol. Listing 6 provides some sample code that establishes a connection to the Amazon SOAP interface and outputs the available methods in a debug window.

Listing 5: XML-Formatted Data

```
01 <forecast>
02   <day label="TODAY"
   desc="Rain Likely"
03 temp="Hi 60&#176;F "/>
04   <day label="Tonight"
   desc="Breezy"
05 temp="Lo 34&#176;F "/>
06   <day label="Wednesday"
   desc="Breezy"
07 temp="Hi 46&#176;F "/>
08    [...]
09 </forecast>
```

SOAP has been criticized for transmitting large volumes of XML code for the simplest of calls. As an alternative to SOAP, OpenLaszlo also supports the simpler XML RPC protocol.

OpenLaszlo is based on Tomcat 5. The OpenLaszlo SDK contains a Tomcat servlet container. The precondition for a full-fledged OpenLaszlo server is a Java Runtime Environment, Version 1.4 or newer. The Java platform brings the ability to access Java objects and methods directly from the client application. An OpenLaszlo server is easy to set up. After installing the Java SDK, unpack the OpenLaszlo tarball and call *Path/to/unpack/directory/server/tomcat-5.0.24/bin/startup.sh*. The sample applications in the archive should then be accessible on *http://localhost:8080/lps-Openlaszlo-Version*. On production servers, a rewrite rule in the Apache configuration ensures that the Apache web server serves up the OpenLaszlo directory as a proxy on the standard port 80:

```
RewriteEngine on
RewriteRule ➥
^/directory(.*)$➥
http://localhost:8080/lps-➥
Openlaszlo-Version$1 [p]
```

Listing 6: SOAP

```
01 <canvas debug="true"
   height="530">
02   <debug x="15" y="15"
   width="415"
03 height="500" />
04   <soap name="amazon"
05       wsdl="http://
   soap.amazon.com/
06 schemas3/AmazonWebServices.
   wsdl">
07    <handler name="onload">
08         Debug.write('Amazon
   soap
09 service loaded');
10         Debug.write('Amazon
   WSDL at ' +
11 this.wsdl);
12         Debug.write('proxy:');
13         Debug.inspect(this.
   proxy);
14    </handler>
15   </soap>
16 </canvas>
```

For more information on server administration, see the OpenLaszlo site [4]. In environments without a Java Runtime Environment, such as shared web hosting without root access, OpenLaszlo can be run in solo mode. An OpenLaszlo server can compile the applications as separate, executable .swx files that provide the whole functionality in Flash code that runs client-side. A web server without Java can provide this support on the Internet; however, RPC functions are not available in these applications.

Conclusions

The OpenLaszlo XML language relies on standards that web developers will be familiar with in many areas. The XML tags for defining display elements are similar to HTML tags. The client-side program logic uses typical DOM syntax to access tags. Javascript 1.4 is used as the programming language. The display elements are visually more attractive than their HTML counterparts and can be animated without too much effort. Besides the typical web programming approach of generating client-side program code in the server's own language, OpenLaszlo has more powerful methods of client-server interaction. In addition to *XMLHttpRequest*, the framework supports SOAP and XML RPC, as well as the option of calling server-side Java program code directly in the client. OpenLaszlo applications run on the proprietary Flashplayer, which might seem to be a disadvantage to some users but guarantees robustness and cross-browser compatibility that removes the need for extensive testing. The next version of OpenLaszlo will be capable of using Ajax for the client-side display and program logic. ■

INFO

[1] Flashplayer use according to the vendor: *http://www.adobe.com/products/player_census/flashplayer/version_penetration.html*

[2] Overview of form elements in OpenLaszlo: *http://www.openlaszlo.org/lps/examples/components/style_example.lzx*

[3] Xpath: *http://en.wikipedia.org/wiki/XPath*

[4] OpenLaszlo server administration: *http://www.openlaszlo.org/lps/docs/deploy/deployers-guide.html*