



United Business Media

PROGRAMMING LANGUAGES

Dr. Dobb's JOURNAL

www.ddj.com The World of Software Development

ISSUE NO. 408
MAY 2008

Software Development Goes to the Movies!

Cat: A Functional Stack-Based Language

KERNEL-MODE DATABASES

Mojax: Mobile Ajax Framework

Getting Better Search Results

Is Perl on Its Way Out? Paul Jansen Thinks So!

PARALLEL EXTENSIONS FOR .NET

Scott W. Ambler:

The Agile Edge

Herb Sutter:

Effective Concurrency

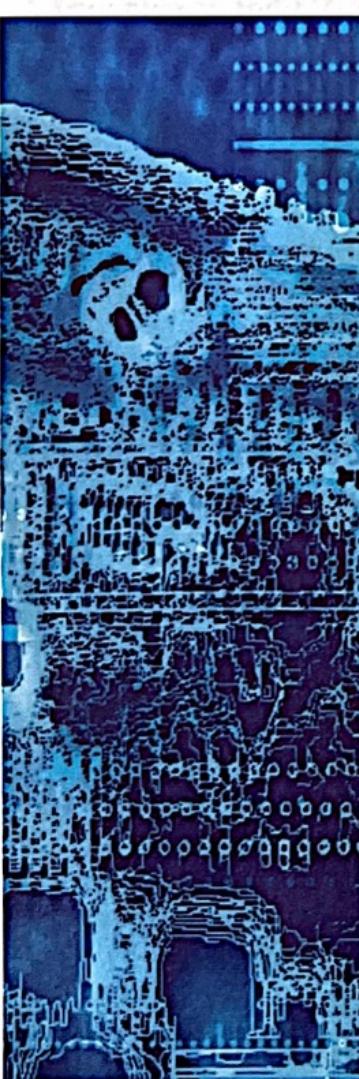
Mojax: Mobile Ajax Framework

The best of Ajax on a mobile platform



John leads developer relations for Mojax. He can be reached at jmuchow@mfoundry.com or visit his blog at 360Mobile.us.

Mojax, short for "Mobile Ajax," is a framework that leverages the technologies that make Ajax a compelling platform for web development (JavaScript, CSS, and XML) and extends them to mobile applications. Mojax applications (mplets) run as native code on a device and are not limited to the constraints of running within a web browser. Mojax applications support development of "plugins" that can access device capabilities such as location services, contacts, audio, and video.



To get started with Mojax development, go to mojax.mfoundry.com. Here you will find documentation, tutorials, and information on how to install an Eclipse plug-in that is designed for developing and testing Mojax applications.

To show how you can use the Mojax framework, I present a sample application that searches for and displays real-estate property information. The entire application consists of a 212-line source-code file, a 78-line style sheet, and a 36-line XSLT filter. (The complete source code is available online; see "Resource Center" page 5.)

The sample application is built around the Zillow.com API (www.zillow.com), which offers a comprehensive set of interfaces for accessing real-estate property information. Figure 1 is the Mojax application I present here. (You get one guess as to whose house this is. Hint: The state and the valuation should give it away.)

When running the application, clicking on the Search option lets you specify the street address, city, and state of the property you want to search for. If the address that you enter is found, the main screen displays the updated property information; if the search fails, a message is returned.

Mojax Programming Model

Like browser-based Ajax applications, the Mojax application UI consists of visual elements arranged

within a screen. The UI elements are defined using Mojax XML tags; for example:

```
<mplet default="main">
  <screen id="main"
    layout="vertical">
    <textbox width="100%"
      halign="center">
      Hello World!</textbox>
  </screen>
</mplet>
```

With the visual elements defined, you specify various attributes such as color, borders, and margins using Cascading Style Sheets (CSS). Example 1 shows one approach for defining CSS content using a `<style>` tag.

The final aspect of developing Mojax applications is to tie together application logic and the interaction among visual elements using Mojax script, an implementation of the ECMAScript-262 Standard (the same standard JavaScript is based on). While the DOM objects available to Mojax developers are unique to the Mojax framework, you will have little trouble with Mojax if you are familiar with HTML DOM.

User Interface

Listing One generates the sample app's splash-screen. There are several things to note about this listing. First, a `<screen>` tag is defined and assigned an ID of `splashScreen`. This identifier refers to this screen when it needs to be shown or hidden on the device display. The layout of contents on the screen

continued from page 30

is a vertical orientation and an `<imagebox>` tag shows a banner across the top of the screen. The remaining content of the splashscreen are within a `<box>` tag, which you can use to further define the layout. The `<textbox>` tag defines the message to be displayed to users while the application loads.

The second `<imagebox>` tag displays an animated progress indicator. Associated with this tag are two `<method>` tags—`onShow` and `onHide`. The former is called

when the imagebox (essentially, the splash-screen) is shown on the device. The work that needs to be done here is to call the `init()` function to initialize the application, start the animation of the progress indicator, and call the Zillow API through the `pingzillow()` method. The `onHide` method is called when this screen is no longer visible on the display, your cue to stop the animation. (Regarding the GIF image: Mojax supports JPG, GIF, and Animated GIF files, even on devices that do not directly support these file types.)

The next screen is the main UI defined using a `<screen>` tag `mainScreen`; see Listing Two. In Listing One, I created two `<method>` tags for managing the `onShow` and `onHide` events. In Listing Two, the events `onLeftSoftkey` and `onRightSoftkey` are defined inline within the `<screen>` tag. Either approach suffices, as the end result is the same; that is, when some specified event occurs, perform some action. Generally, I use the `<method>` tag when more than one action is tied to an event.

There is a powerful concept hidden within the `<textbox>` tags—the ability to work with dynamic content using `bind`, which associates an expression with the value of an object. In the two textboxes shown, `bind` displays the current address, city, and state by retrieving the current value of `Cache.address` and `Cache.citystate`.

The primary content on the main screen is contained with a `<scrollbox>` tag. I use this tag when the contents of the property information returned from Zillow can't be shown on one screen. The contents of the scrollbox are managed in the `<propertydetail>` tag.

Notice the softkey references (`onLeftSoftkey/onRightSoftkey`) and the methods associated with them. When users select the left softkey, `exit()` is called to shut down the application. When the right softkey is pressed, the method `show()` is invoked,

Intellectuals solve problems.
Geniuses prevent them.

— Albert Einstein

ADOPT A CODING STANDARD
AUTOMATICALLY ENFORCE IT
MEASURE COMPLIANCE

RESOURCES:
High-Integrity C++
MISRA-C
JSF AV C++
Your Best Practices

"Solutions" that
only identify problems
are simply not the solution.

THINK PREVENTION
GET QUALITY

www.PROGRAMMINGRESEARCH.COM

 PRQA
Programming Research
THE CODING STANDARD EXPERTS



Figure 1: Main screen.

passing in the name of the screen to show; in this case, the search screen.

Styling with CSS

You can use style sheets within Mojax to define colors, fonts, layout, and other visual aspects of the application. Style sheets separate style elements of an application from the actual content. Style-sheet information can be specified in many contexts. Looking at the definition for mainScreen (Listing Two), you notice that style attributes are defined inline within each of the two textboxes. An additional option is to define style information using a `<style>` tag:

```
<style>
  .splashtext {
    color: #00008B;
    background-color: #FFFFFF;
    font-size: large;
    font-weight: bold;
    font-family: system;
  }
</style>
```

The preferred means for defining style information is to place all definitions within a file. For example, the style information for this application is stored within a file named "mZillow.mcss." The code below is a partial listing of the style information.

```
textinput {
  color: #00008B;
  ...
}

textinput:focus {
  border-width: 1px;
  border-color: #00008B;
  ...
}
```

In this style definition I specify the look of the `<textinput>` tag. Pay attention to the differences in the attributes for a `textinput` box and a `textinput` box with focus. With this approach, additional code is not needed in the application to check whether a `textinput` box has focus. Instead, with style sheets, such changes are managed for you by the Mojax runtime. The aforementioned style attributes are used within the search screen when prompting for an address, city, and state, which is shown in Example 2.

Example 3 is an illustration of accessing style information through class attributes within a tag. Notice in Figure 1 that a grid is displayed with two columns and six rows. Class attributes are used within the

```
<moblet default="main">
  <style>
    screen {
      color: #483D8B;
      background-color: #DCDCDC;
      font-size: medium;
    }
  </style>
  <screen id="main" layout="vertical">
    <textbox width="100%" halign="center">Hello World!</textbox>
  </screen>
</moblet>
```

Example 1: Defining CSS content.

```
<screen id="searchScreen" layout="vertical">
  <imagebox url="Images/zillow.gif" width="100%" halign="center"/>
  <box height="100%" width="100%" layout="vertical">
    <!-- Two textboxes for input -->
    <textbox style="padding: 2px;">Enter street address:</textbox>
    <textinput length="25" value="bind{Cache.address}" />
    <textbox style="padding: 2px;">Enter city and state:</textbox>
    <textinput length="25" value="bind{Cache.citystate}" />
  </box>
  ...
</screen>
```

Example 2: Search screen code (partial).

```
.propertylabel {
  background-color: #DAE4F6;
  padding: 2px;
  color: #445681;
  border-bottom-width: 1px;
  border-bottom-color: #445681;
}
<prototype name="propertydetail" layout="vertical" extends="Box" width="100%">
  <gridbox cols="2" rows="6" height="100%" width="100%" layout="horizontal">
    <align="top">
      <textbox width="100%" class="propertylabel">Year Built:</textbox>
      <textbox width="100%" class="propertyvalue">bind{propertyXML[0].yearbuilt}</textbox>
    ...
  </gridbox>
</prototype>
```

Example 3: Accessing class attributes within a tag.

```
<script><![CDATA[
  function init()
  {
    // If no cached value
    if (!Cache.address)
    {
      // Default to zillow example address
      Cache.address = "2114 Bigelow Ave";
      Cache.citystate = "Seattle,WA";
    }
    ...
  ]]></script>
```

Example 4: Using Mojax Script.

```
<prototype name="softkeys" extends="Box" layout="horizontal" width="100%">
  <attribute name="left">
    this.lefttext.value = this.left;
    this.lefttext.visible = true;
  </attribute>
  <attribute name="right">
    this.righttext.value = this.right;
    this.righttext.visible = true;
  </attribute>
  <textbox id="lefttext" width="100%" halign="left" visible="false"/>
  <textbox id="righttext" width="100%" halign="right" visible="false"/>
</prototype>
```

Example 5: Extending the Box object.

Listing One

```
<screen id="splashScreen" layout="vertical">
  <imagebox url="Images/zillow.gif" width="100%" halign="center"/>
  <box height="100%" width="100%" layout="vertical" halign="center"
    valign="center">
    <textbox class="splashtext" width="100%" halign="center" valign="center">
      Loading, please wait ...
    </textbox>
    <imagebox id="splashloading" focusable="false" url="Images/loading.gif">
      <method name="onShow">
        init();                                // Initiation application data
        this.animate(true);                     // Start animation
        pingzillow();                          // Get property information
      </method>
      <method name="onHide">
        this.animate(false);                  // Stop animation
      </method>
    </imagebox>
  </box>
</screen>
```

Listing Two

```
<screen id="mainScreen" layout="vertical" onLeftSoftkey="exit()"
  onRightSoftkey="show(searchScreen)">
  <imagebox url="Images/zillow.gif" width="100%" halign="center"/>
  <!-- Show address info across top -->
  <textbox layout="vertical" width="100%" halign="center" style="padding: 1px;">
    valign="center" value="bind{Cache.address}" />
  <textbox layout="vertical" width="100%" halign="center" style="padding: 1px; border-bottom-width: 1px; border-bottom-color: #00008B;">
    valign="center" value="bind{Cache.citystate}" />
  <!-- Property information -->
  <scrollbox width="100%" height="100%" focusable="true" scrollbar="true" >
    <propertydetail/>
  </scrollbox>
  <softkeys left="Exit" right="Search"/>
</screen>
```

<textbox> tags to specify the background color, padding, and other attributes. With a few quick changes to the style sheet, you can quickly change the look-and-feel of the application.

Mojax Script

Mojax provides an implementation of ECMAScript, not unlike ActionScript (Adobe) or JScript (Microsoft). Example 4 illustrates it being used for an initialization function.

Another common use of scripting is within a **<method>** tag. For example, when processing key events, you can opt to check whether a numeric key was pressed:

```
<method name="onKeyPressed">
  if (event.isNumeric() == true)
  {
    ...
  }
</method>
```

The uses of script within Mojax are many and varied. For instance, you could embed a short script within a **<screen>** tag as follows to display a debug message to the development environment console when the user selects the right softkey:

```
<screen id="mainScreen" layout="vertical"
  onRightSoftkey="if (varx > 10)
    debug('varx: ' + varx);">
  ...
</screen>
```

Data Persistence

The **Cache** is a globally accessible object that provides a means to persist data across invocations of a mobile application. The **Cache** object is a collection of properties, stored as name-values pairs. Properties of the **Cache** are set and retrieved using dot notation:

```
Cache.address = "2114 Bigelow Ave";
```

The **Cache** is limited by available storage space on the device, so it's important to be judicious in deciding what to cache. Also, caching is on a per moblet basis; that is, each moblet has its own **Cache** so there are no worries about one moblet overwriting another's cached content.

Preloading and Caching

Mojax can preload images during application startup via the **<resource>** tag:

Maintain Your Software with Understand™

You didn't write that mangled code your company picked up in a merger. But it's big, it's ugly, and you need help.

Help is here! Understand helps software developers maintain, measure, document and comprehend source code they didn't write, wrote long ago, or is too large and complex.

What makes us better? Understand quickly parses and manipulates very large amounts of code. One million SLOC projects are common with our users. We also focus on exceptional customer support with rapid response from real engineers. New features and fixes are incorporated into weekly builds.

- Works with K&R C, ANSI C, C++, Java, Ada, FORTRAN, Pascal, Jovial
- Fast on really big projects
- Great for exploring code
- Detailed x-referencing & 1-click visiting
- Info on everything called/used in code
- Works on partial code or missing pre-processor definitions
- Class browsing (inheritance & base)
- Easy generation of tree graphs
- Visio export of graphics
- PERL or C API to get data/graphics for custom documentation
- Comment extraction for documentation generation
- Freely sharable HTML and text reports of analysis information
- Complexity, Size, OO, other metrics
- Easy export of metrics to spreadsheets and databases

FREE Trial Version at www.scitools.com 435.627.2529

Scientific Toolworks Inc.
Maintain Your Software

```
<resource url="Images/zillow.gif" persist="true"/>
```

The *persist* property is optional in the *<resource>* tag. However, the advantage of setting *persist* to *true* is that once the image is downloaded, it is stored on the device; thus, on startup of the application next time around, the image is available.

HTTP, XML, and XSLT

Given the nature of mobile devices and their ability to communicate over the air, many Mojax applications access remote resources. The *<httprequest>* tag is for content types accessible via HTTP. Listing Four (available online) presents the *httprequest* for the sample application. The *url* property is set to match the format for calling the Zillow property details API. The *async* property is set to *true*, enabling the request to be completed in the background.

I apply an XSLT filter (Listing Five; available online) to the XML response to tailor the data specifically for the application. A side-effect of using the filter is reducing network traffic to only the information needed, thus improving application performance.

The bulk of the interesting code regarding the HTTP request is inside the *<method>* tag *onLoaded*, which is called once the request has completed. The response from the HTTP request is used to create two XPath expressions for accessing the returned XML.

For completeness, it's important to note that Mojax currently supports XML, JSON, images, and Mojax script as the supported data types through an HTTP request.

Creating Objects Using Prototypes

Prototypes are a powerful concept for encapsulating common code into reusable classes. Within Mojax, the *<prototype>* tag is the equivalent of the *prototype* object in ECMAScript. One common use of a prototype within Mojax apps is for managing softkeys. Notice in Example 5 how the prototype extends the *Box* object, which is a logical choice as this object is designed specifically as a container for other visual

elements. Attributes are defined, one for each softkey, left and right. Two *<textbox>* tags are defined, one for each label to be associated with the left and right keys. Once the prototype is defined, you can use the object as you would any other Mojax tag. For example, the code below shows a *<screen>* tag definition that declares a *<softkeys>* tag used to associate the text "Exit" with the left softkey, and the text "Search" with the right.

```
<screen id="mainScreen" layout="vertical" onLeftSoftkey="exit()" onRightSoftkey="show(searchScreen)">...<softkeys left="Exit" right="Search"/></screen>
```

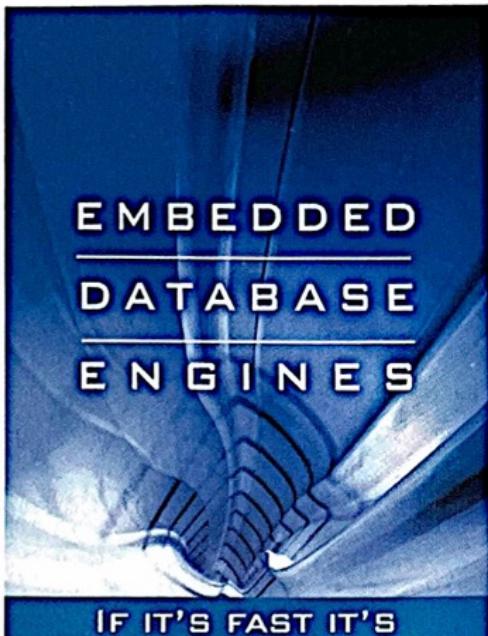
Referring back to Example 5, notice that inside the *<attribute>* tag I referenced the *<textbox>* tags using the IDs assigned to the textbox tags; for example, *this.lefttext.value*. When you define a *<softkeys>* tag, an event is generated for each attribute defined in the tag. For example, *left="Exit"* generates a call to the attribute tag within the prototype definition with the name *left*, which sets the value of the first textbox to *Exit* and sets its visible property to *true*. The same logic applies for the *right="Search"* definition, setting the value and visible properties of the object.

The reason for the visible property in the prototype definition is to allow use of a *<softkey>* tag, which has just one key definition. In other words, you may want to enable just one softkey for a screen:

```
<screen id="anotherScreen" layout="vertical" onLeftSoftkey="show(mainScreen)">...<softkeys left="Back"/></screen>
```

By definition, the textboxes in the prototype are not visible; thus, just one label (Back) is shown on the device display.

In Listing Two, you see the reference to a *<propertydetail>* tag. Figure 1 shows the device output when using this tag, which displays the property information returned for the Zillow API. In Listing Four (available online) the variable *propertyXML* was assigned the value of the XPath expression:



Highest Performance Disk and In-memory Embedded Database

Fault tolerant and un-matched reliability with both data replication and mirroring.

Solve complex data management problems through the network model.

Full interoperability with backend systems through SQL.

80+ supported platforms, and we'll add yours!

Reduce costs with our extensive development tools and utilities.

Multiple products allow future scalability without change.

Resource constrained to enterprise applications.

Highest quality products with 25 years of successful deployment.

Download a FREE development kit and see what 20,000 other developers rave about!

birdstep
TECHNOLOGY

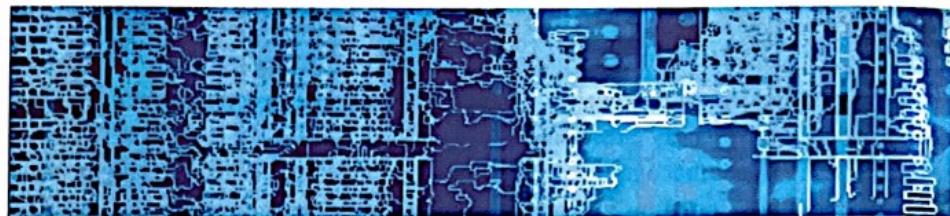
<http://www.birdstep.com/database>
americas@birdstep.com
+1 206 748 5353

```
propertyXML =
    xpath(response#/response/result);
```

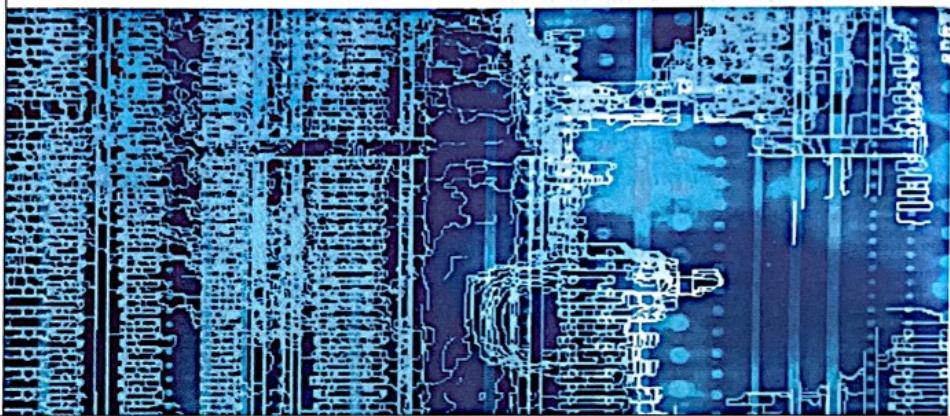
In the `<propertydetail>` tag, this variable is accessed to dynamically bind the results of the XML response to the textboxes on the main screen. Using *bind*, the content of the screen is automatically bound to the current value of the XML response, which allows the application to dynamically update its contents based on the results returned from Zillow.

Conclusion

Mojax extends the concepts that drive Ajax technology to mobile applications. Using JavaScript, CSS, and XML/XSLT, you can develop mobile applications for a range of devices and platforms, with one code base. In approximately 325 lines of code, I created a cross-platform application that accesses a remote API over HTTP and dynamically binds the returned information to visual elements on a mobile device display. **DDJ**



Mojax extends the concepts that drive Ajax technology to mobile applications



I/O Controls

- High-Speed for Real-Time applications
- Built-In custom Property Editors
- Automatic and Custom Sizing. No Restrictive Bitmaps
- Look and Feel of Real Hardware
- Includes : Switches, Gauges, Sliders, Led's, Led Bar, Led Spiral, Integer/Binary/Hexadecimal Displays, Tanks, Valves, Motors, LCD Matrix, Spectrum Display, Percent and Pie Graph, Odometers, Analog Clock, Image Display, Rotation Display, and Mode Combo Box.

Std Pack

- 28 Controls
- Basic I/O Controls

Single Developer : \$559
Additional Developer : \$189

Pro Pack

- 55 Controls
- Basic & Advanced I/O Controls

Single Developer : \$1099
Additional Developer : \$379

Plot Pack

- Plotting : Scientific, Engineering, Strip-Chart, Digital, EKG, and more...

Single Developer : \$859
Additional Developer : \$289

Ultra Pack

- 56 Controls
- Basic & Advanced I/O Controls plus Plotting

Single Developer : \$1699
Additional Developer : \$579