Getting Started With Project jMaki for the GlassFish v3 Application Server

Technology Preview 2

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–4868–05
June 2008

080606@20209

Contents

Getting Started with Project jMaki 5
Introduction to Project jMaki 5
Obtaining Required Software 6
▼ Installing the GlassFish v3 JavaEE Integration Plug-in and the jMaki Ajax Support Plug-in 6
The `plotCity` Example 7
What the `plotCity` Example Does 7
Downloading and Running the Example 7
Creating a jMaki Web Application 8
▼ Creating a jMaki Web Application 8
Adding a jMaki Widget to a Page 9
Loading Your Own Data Into a jMaki Widget 11
Initializing the Widget With Data 11
Updating the Data in a Widget 12
Handling Widget Events 13
Accessing an External Service From a Widget 15
Developing the `plotCity` Application Using Project jMaki and the NetBeans IDE 6.1 18
▼ Creating the `plotCity` Project 18
▼ Adding the Widgets to the Page 18
Loading Data Into a Widget in the `plotCity` Application 21
Handling Events in the `plotCity` Application 24
▼ Publishing to a Topic 25
▼ Writing a Handler that Handles the Widget Event 25
▼ Writing a Servlet to Obtain the Server-Side Data for the Handler 26
▼ Subscribe to a Topic 27
▼ Accessing an External Service from a Widget 27
▼ Deploying and Running on the GlassFish v3 Technology Preview 2 Application Server ... 28

Index 31

Getting Started with Project jMaki

This tutorial helps you get started using Project jMaki with the GlassFish™ v3 Technology Preview 2 Application Server by covering the following topics:

- Obtaining Required Software
- Creating a jMaki Application
- Adding Data to a Widget
- Handling Events
- Accessing External Services

Introduction to Project jMaki

Project jMaki is a lightweight framework for creating web applications using built-in templates, a model for creating and using Ajax-enabled widgets, and a set of services to tie the widgets together and enable them to communicate with external services.

Project jMaki provides a set of pre-wrapped widgets, many of which are from Dojo, script.acu.lous, Yahoo UI, and other vendors. Because the widgets are wrapped, you can use them in a variety of server environments, including as JavaServer Pages™ (JSP™) tags , as JavaServer™ Faces components, within a Phobos application, with JRuby, or with PHP. This chapter focuses on using jMaki in a JSP application for the GlassFish v3 application server.

In addition to the set of widgets, jMaki also provides the following features:

- A way to easily customize a widget, such as set the value of its attributes and load your own data into the widget
- Mechanisms for responding to widget events, getting widgets to interact, and allowing widgets access to external services.
- A NetBeans IDE 6.1 module that enables you to quickly and easily create web applications with jMaki.

Obtaining Required Software

This tutorial uses The NetBeans™ IDE 6.1 to develop and deploy a jMaki application. The IDE offers a plug-in through its update center that allows you to quickly and easily create jMaki applications.

It also offers a plug-in that allows you to add the GlassFish v3 Technology Preview 2 Application Server to your list of servers supported by the IDE. In addition to the jMaki module and the application server, you also need to download a web browser that supports JavaScript and Ajax.

▼ Installing the GlassFish v3 JavaEE Integration Plug-in and the jMaki Ajax Support Plug-in

With this task, you will install the GlassFish v3 Java EE Integration plug-in and the jMaki plug-in into NetBeans IDE 6.1.

1. **Within the IDE, select Tools from the menu bar.**
2. **Select Plugins.**
3. **Select the Settings tab.**
4. **Select the Available Plugins pane.**
5. **Select the checkbox next to GlassFish v3 JavaEE Integration Module.**
6. **Also on the Available Plugins pane, select the checkbox next to jMaki Ajax Support.**
7. **Click Install.**
8. **Select Restart the IDE Now.**
9. **After the IDE restarts, go to the Tools menu and select Servers.**
10. **Click Add Server.**
11. **Select GlassFish v3 TP2 from the Server pane.**
12. **Enter a name for the server in the Name field and click Next.**
13. **Click Browse to select the installation location for the server.**
14. **Select the checkbox to indicate you've read the license and click Download V3 Now.**

The plotCity Example

This section describes the plotCity example and tells you how to run it.

What the plotCity Example Does

With the plotCity example, you can plot a city on a map by performing the following tasks:

1. Select a state from a Dojo combobox widget, thereby populating another Dojo combobox widget with a list of cities located in that state.
2. Select a city from the second Dojo combobox widget, thereby causing the application to access the city's coordinates from an external service and plot the city on a map represented by a Google map widget.

This tutorial uses the plotCity example to demonstrate using the primary features of jMaki:

- An extensive widget tag library for use in JSP pages.
- Easy access to server—side data from jMaki widgets.
- Mechanisms to perform widget-to-widget communication.
- A proxy service to access external services.

Downloading and Running the Example

This section tells you how to get the example, build it, and run it.

▼ Downloading the Example

1 **Go to the jMaki sample WAR files download area .**

2 **Download one of the following files:**

- `plotCity.war`**: Download this file if you just want to deploy and run the example the way you do for any other web application.**
- `plotCity.zip`**: Download this file if you want to load the plotCity project in NetBeans IDE 6.1, build it, deploy it, and run it.**

3 **If you downloaded the** `plotCity.zip` **file, extract the zip file.**

▼ Building and Running the plotCity Application in NetBeans IDE 6.1

With this task, you'll be able to run plotCity in your browser.

1 **Select File→Open Project in NetBeans IDE 6.1.**

2. **Select the `plotCity` project that you extracted from the zip file in the previous task, and click OK.**

3. **Right click on the `plotCity` application in the Projects pane and select Run Project.**

   This will compile the classes, package the files into a WAR file, deploy the WAR to your GlassFish v3 Technology Preview 2 Application server instance, and open a web browser to the following URL:

   ```
   http://<server>:<server port>/plotCity/
   ```

4. **Select a state from the State combobox. Then select a city from the City combobox. You'll see that city plotted on the map.**

Creating a jMaki Web Application

The NetBeans 6.1 IDE jMaki 1.1 plug-in allows you to quickly and easily create jMaki applications.

With the plug-in, you can drag and drop jMaki widgets onto JSP pages and customize the widgets. When you create a page using the plug-in, you have a choice of templates to use for the page. You can read about the templates and see what they look like by visiting the jMaki Layouts page.

▼ Creating a jMaki Web Application

After you complete the following task, the IDE creates a web application with the necessary resources. The Web Pages node contains the following items:

- `WEB-INF`. This directory contains the project deployment descriptor files.
- resources node. This node contains the jMaki JavaScript files and component files for using widgets in your application.
- `index.jsp`. This is your welcome JSP page.
- `glue.js`. You can add event handling functions to this file.

The IDE opens the `index.jsp` page in the editor pane after you create the application.

1. **Choose File > New Project.**

2. **Under Categories, select Web.**

3. **Under Projects, select Web Application and click Next.**

4. **Type the name of your application in the Project Name field.**

5. **Change the Project Location to any directory on your computer.**
6. **Select GlassFish V3 TP2 for the server.**
7. **Keep the other settings at their default and Click Next.**
8. **Select the jMaki Ajax Framework and select a template from the CSS layout pane.**
9. **Click Finish.**

Adding a jMaki Widget to a Page

Once you have set up your application and created a page using one of the templates, you can add a jMaki widget to the page. "Developing the `plotCity` Application Using Project jMaki and the NetBeans IDE 6.1" on page 18 details how to build a jMaki application using the NetBeans IDE 6.1 plug-in, using `plotCity` as an example. When you drag and drop a widget on a page using the plug-in, it does three things:

- Adds the widget resources to the application.
- Adds the jMaki tag library definition to the page.
- Adds to the page a custom jMaki `widget` tag that references the widget and sets the widgets attributes to default values.

For example, if you added the jMaki Dojo combobox widget to a page, the IDE does the following:

1. Adds the `component.js` and `component.htm` files to the `resources/dojo` directory of your application
2. Adds the third-party Dojo widget code to the `resources/libs` directory of your application.
3. Adds the following tag library declaration and `widget` tag to your page:

```
<%@ taglib prefix="a" uri="http://jmaki/v1.0/jsp" %>
...
<a:widget name="dojo.combobox"
   value="[
       {label : 'Alabama', value : 'AL'},
       {label : 'California', value : 'CA'},
       {label : 'New York', value : 'NY', selected : true},
       {label : 'Texas', value : 'TX'}
    ]" />
```

As the preceding code shows, the `widget` tag is a custom tag from the jMaki tag library. This tag represents a JSP tag handler.

When you drag and drop the combobox widget onto a page, the IDE initializes the tag with the name of the widget using the `name` attribute. The name is in dot notation similar to Java™ package names. It refers to the directory that contains the widget's resource files. In this case, the combobox widget's resources are in the `dojo/combobox` directory.

The IDE also initializes the widget with a default value using the `value` attribute. As most jMaki widgets do, the combobox widget accepts data in JSON format. You can use the `value` attribute to reference this data. For example, the default value that NetBeans gives the combobox widget is:

```
[
{label : 'Alabama', value : 'AL'},
{label : 'California', value : 'CA'},
{label : 'New York', value : 'NY', selected : true},
{label : 'Texas', value : 'TX'}
]
```

The next section details how to populate a widget with your own data.

The `widget` tag requires you to set a different set of attributes depending on which widget the tag represents. The best way to determine what attributes a widget expects is to look at the widget's `widget.json` file by right-clicking on the `widget` tag in the IDE and selecting jMaki from the pop-up menu. You can use the customizer that pops up to edit the values of the attributes.

The following table lists the most common attributes.

TABLE 1 Common jMaki Attributes

Attribute	Required	Value
id	No	Identifies the widget instance so that you can reference the widget.
name	Yes	The name of the widget
style	No	The CSS style for this widget. Defaults to `component.css`
service	No	Refers to a component that provides extra services for the widget. You can use this attribute to reference a component that serves data to the widget.
value	No	References the widget's data
args	No	object literal that defines additional tag attributes

For the `plotCity` example, you also need to add `id` attributes to the combobox widget tags so that you can refer to the widgets from your event handlers, as explained in "Handling Widget

Events" on page 13. You'll name the combobox that holds the states `thisState`, and you'll name the combobox that holds the cities `thisCity`:

```
<a:widget  id="thisState"
    name="dojo.combobox"
    ... />
<a:widget id="thisCity"
    name="dojo.combobox"
    ... />
```

Loading Your Own Data Into a jMaki Widget

Project jMaki gives you a lot of flexibility with respect to how you populate your widgets with data. Following are the three basic techniques for loading data into your widget:

- Reference a static file that contains the JSON data.
- Use an expression language (EL) expression expression from the tag's `value` attribute to reference the data from a bean.
- Use the tag's `service` attribute to reference data served by a JSP page or a servlet.

This section shows you how to use an EL expression from the `value` attribute to access data from a bean, just as you would with any other JSP tag. Whichever method you choose, you need to be sure that you pass the data in JSON format to the widget because this is what all the jMaki widgets expect. This means that if you have the data in a bean, you need to convert it from a Java object to JSON. jMaki includes the JSON API, which you use to perform the data conversion.

Initializing the Widget With Data

The `thisState` and `thisCity` combobox widgets on the `index.jsp` page use EL expressions to get their initial set of data from `StateBean`. The following code snippet shows the `jsp:useBean` and `widget` tags from this page:

```
<jsp:useBean id="StateBean" scope="session"
    class="plotCity.StateBean" />
<a:widget id="thisState" name="dojo.combobox" ...
    value="${StateBean.states}"/>
<a:widget id="thisCity" name="dojo.combobox" ...
    value="${StateBean.cities}" />
```

As the preceding markup shows, the data comes from the `getStates` method of `StateBean`. The `useBean` tag makes `StateBean` available to the `index.jsp` page.

In `StateBean`, the `getStates` method converts string arrays from Java code into a single JSON array:

```
public String getStates() throws JSONException {
    JSONArray statesData = new JSONArray();
    JSONObject stateData = new JSONObject();
    for (int loop = 0; loop < states.length; loop++) {
        stateData.put("label", states[loop]);
        stateData.put("value", stateCodes[loop]);
        statesData.put(stateData);
        stateData = new JSONObject();
    }
    return jmaki.util.JSONUtil.jsonArrayToString(statesData, new StringBuffer());
}
```

The `getStates` method uses the JSONArray API to create the following JSON array:

```
"[
    {label : 'Alaska', value : 'AK'},
    {label : 'Arizona', value : 'AZ'},
    {label : 'California', value : 'CA'},
    {label : 'Oregon', value : 'OR'},
    {label : 'Washington', value : 'WA'}
]"
```

Using the `${StateBean.states}` expression, the combobox widget loads this JSON array.

The `getCities` method loads all the Alaskan cities into a JSON array because Alaska is the initially selected state.

Both methods call the `jsonArrayToString` method to convert a JSON array to a string before returning the data back to the page.

It's important to remember that the EL expressions can only be used to initialize the widgets with data. To update the widget's data, such as after a user-initiated event, you need to use the publish/subscribe mechanism, as described in the section, "Handling Widget Events" on page 13.

Updating the Data in a Widget

In the case of the `plotCity` example, when you use the publish/subscribe system to update the widget's data, you are still using `StateBean` to get the data, which is stored in the `cities.properties` file.

Here are the contents of the `cities.properties` file:

```
AK=Anchorage,Fairbanks,Juneau,Nome
AZ=Mesa,Phoenix,Scottsdale,Tucson
CA=Los Angeles,Sacramento,San Diego,San Francisco
OR=Bend,Eugene,Portland,Salem
WA=Olympia,Seattle,Spokane,Tacoma
```

The event handler, as described in the next section, uses Ajax to send the selected state value to the `StateBean` object's `getNewCities` method, by way of a servlet. The `getNewCities` method gets the list of cities for the selected state from the properties file and loads the cities into a JSON array, as shown in the following code:

```
...
cityNames = ResourceBundle.getBundle("plotCity.cities");
...
public String getNewCities(String state) throws JSONException{
    JSONObject city = new JSONObject();
    JSONArray cities = new JSONArray();
    String[] names = null;
    try {
        names = cityNames.getString(state).split(",");
    } catch(Exception e){
        return null;
    }
    for(int i = 0; i < names.length; i++){
        city.put("label", names[i]);
        city.put("value", names[i]);
        cities.put(city);
        city = new JSONObject();
    }
    jmaki.util.JSONUtil.jsonArrayToString(cities, new StringBuffer());
}
```

"Loading Data Into a Widget in the `plotCity` Application" on page 21gives step-by-step instructions for implementing `plotCity` to populate its widgets with data.

Handling Widget Events

Project jMaki supports a topic-based publish/subscribe system in which widgets can act as producers or consumers. In this system, producers publish messages to topics and consumers receive the messages from topics to which they have subscribed.

This system makes it easy for your application to respond to widget events and to get two widgets to interact by listening for each other's events. This section describes how you can get widgets to publish and subscribe to a topic so that your application can respond to a widget event.

To make event-handling easier, jMaki provides default topics for the most common widget events. For example, a Dojo combobox widget will publish its value to the `/onSelect` topic when the user selects a value from the widget. Likewise, a Dojo combobox widget automatically subscribes to the `/setValues` topic, and so it gets its value from there. Most often, though, you will need to create your own topics, as the `plotCity` example does.

Usually, you go through the following steps to handle a widget event using the publish and subscribe system:

1. Add a `publish` attribute to a widget tag and set it to a topic string so that the widget will publish its current value to that topic when it experiences an event.
2. Write a handler in `glue.js` that will be called when the widget experiences an event. When the handler is called, it retrieves the value from the topic, performs some task with the value, and publishes the result of the task to a different topic.
3. Add a `subscribe` attribute to another widget tag and set the `subscribe` attribute to the topic to which the handler publishes the result of its task.

In addition to the publish/subscribe mechanism, jMaki provides the `doAjax` function to make it easy to use Ajax within your event handler function to asynchronously retrieve data from the server and return it to the client.

The `plotCity` application uses the publish/subscribe mechanism and `doAjax` so that the following happens:

- When the user selects a state from the `thisState` combobox, the `thisCity` combobox repopulates with the list of cities located in that state.
- When the user selects a city from the `thisCity` combobox, the city is plotted on the map.

In the `plotCity` example, the `thisState` widget's `publish` attribute is set to the topic, `/cb/getState`:

```
<a:widget  id="thisState"
    name="dojo.combobox"
    publish="/cb/getState"
    value="${StateBean.states}" />
```

This makes the widget publish its selected value to the developer-defined `/cb/getState` topic whenever the user selects a value from the widget.

The following event handler in the `glue.js` file subscribes to the `/cb/getState` topic:

```
jmaki.subscribe("/cb/getState/*", function(args) {
    var message = args.value;
    jmaki.doAjax({method: "POST",
    url: "Service?message=" + encodeURIComponent(message),
    callback: function(_req) {
        var tmp = _req.responseText;
        var obj = eval("(" + tmp + ")");
        jmaki.publish('/cb/setValues', obj);
        // handle any errors
        }
    });
});
```

Because the handler subscribes to the topic, it executes when the widget publishes a new value to the topic. The handler does the following:

- Subscribes to the topic, `/cb/getState/*`. Because of the wildcard character, the `subscribe` function subscribes to all subtopics of the `/cb/getState/` topic.
- Receives the state code as an `args` variable from the `/cb/getState` topic. The `thisState` widget publishes to `/cb/getState`, and therefore passes its `args` argument to it when the widget experiences an `onSelect` event.
- Uses the `jmaki.doAjax` function to pass the state code to the `Service` servlet, retrieves the list of cities from the servlet, and publishes the list to the `/cb/setValue` topic. "Writing a Servlet to Obtain the Server-Side Data for the Handler" on page 26 gives more detail on the `Service` servlet.

All combobox widgets automatically obtain their values in response to a user action by subscribing to the global `/setValues` topic that jMaki provides unless the developer specifies otherwise. In this example, `thisCity` subscribes to a `/setValues` subtopic of the developer-defined parent topic, `cb`. Because the `thisCity` combobox subscribes to the parent topic, it also subscribes to the `/setValues` subtopic by default and is therefore automatically updated with the new set of cities.

If you have only one combobox in a page, you do not have to create a parent topic for the purpose of updating values in the combobox. Instead, you can publish the new values to the global `/setValues` topic. This example needs a parent topic because it has two comboboxes. If the handler published to the global `/setValues` topic, both comboboxes would be populated with the set of cities returned by the `Service` servlet.

"Handling Events in the `plotCity` Application" on page 24 gives the step-by-step instructions for implementing event handling in `plotCity`.

Accessing an External Service From a Widget

One characteristic of an Ajax-based client is that it cannot make calls to URLs outside of its domain, which means that it cannot access services located on another server. Project jMaki provides a proxy, called the `XmlHttpProxy` client, that communicates with external services on a widget's behalf.

To access an external service, a widget uses an `XmlHttpRequest` object to access the service through the `XmlHttpProxy` client. Project jMaki already includes some code that allows you to access the Yahoo Geocoder service. This service takes a location, such as a city, and returns the coordinates for that location. The code jMaki provides for widgets to use this service includes the configuration of the service in the centralized `xhp.json` file:

```
{"xhp": {
    "version": "1.0",
    "services": [
```

```
        {"id": "yahoogeocoder",
         "url":"http://api.local.yahoo.com/MapsService/V1/geocode",
         "apikey" : "appid=jmaki-key",
         "xslStyleSheet": "yahoo-geocoder.xsl",
         "defaultURLParams": "location=santa+clara,+ca"
        },
    ...
]}}
```

The preceding JSON code configures the Yahoo Geocoder service under the ID `yahoogeocoder` with the `XmlHttpProxy` client. The configuration includes the URL to the service, a reference to the API key to use for Yahoo services and widgets, a reference to a stylesheet that styles the data returned from the service, and a default location.

To obtain coordinates for a location from the Yahoo Geocoder service from your glue code, you create a string URL out of these elements:

- `jmaki.xhp`, which is a map to the `XMLHttpProxy` service
- The string, `?id=yahoogeocoder&urlparams=`, which combines the ID of the Yahoo Geocoder service with the a set of parameters to pass to it.
- A variable representing the location for which you want to obtain coordinates.

An example URL constructed from this string would be the following:

```
http://localhost:8080/jmaki/xhp?key=yahoogeocoder&urlparams=location=Eugene,Oregon
```

Once you have the string URL, you use Ajax to do the following:

1. Access the service with the URL.
2. Obtain the coordinates from the service.
3. Publish the coordinates to the `/jmaki/plotmap` topic.

The first thing to do is to get the location so that you can provide it to the service. In the `plotCity` application, the `thisCity` widget publishes the city to be plotted to the `/cities` topic when the user selects a city:

```
<a:widget id="thisCity"
    name="dojo.combobox"
    publish="/cities"
    subscribe="/cb"
    value="${StateBean.cities}" />
```

A combobox widget always publishes its selected value to the global `onSelect` topic and to the `onSelect` sub-topic of any developer-defined parent topic. In this case, the combobox widget publishes its value to the parent topic, `/cities`. Therefore, the value is published to `/cities/onSelect`.

Here is the handler from the `plotCity` example that subscribes to the `/cities/onSelect` topic:

```
jmaki.subscribe("/cities/onSelect", function(item) {
    var city = item.value;
    var state = jmaki.attributes.get('thisState').getValue();
    var location = city + ", " + state;
    var encodedLocation = encodeURIComponent("location=" + location);
    var url = jmaki.xhp +
        "?id=yahoogeocoder&urlparams=" +
        encodedLocation;
    jmaki.doAjax({url: url, callback : function(req) {
        if (req.responseText.length > 0) {
            // convert the response to an object
            var response = eval("(" + req.responseText + ")");
            var coordinates = response.coordinates;
            v = {results:coordinates};
            jmaki.publish("/jmaki/plotmap", coordinates);
        } else {
            jmaki.log("Failed to get coordinates for " +
                location );
        }
    }
});
```

This handler does the following:

1. Gets the city name from the `args` argument that is passed to it from the topic, `/cities/onSelect`.
2. Gets the current state from `thisState` and appends it to the city to construct the `location` argument.
3. Calls `encodeURIComponent` on the `location` variable to encode it so that it is the proper format for the URL.
4. Constructs the URL to pass to access the service.
5. Uses Ajax to pass the location to the service, retrieve the coordinates from the service, and publish the coordinates to the `/jmaki/plotmap` topic.

All widgets automatically subscribe to the `/jmaki/plotmap` topic. Therefore, when new coordinates are posted to the topic, the Google map widget plots the city located at the coordinates.

Developing the plotCity Application Using Project jMaki and the NetBeans IDE 6.1

In this exercise you create the plotCity web application using the NetBeans IDE 6.1.

▼ Creating the plotCity Project

When you complete this task, you'll have a skeleton jMaki application.

1. **Choose File > New Project.**
2. **Under Categories, select Web.**
3. **Under Projects, select Web Application and click Next.**
4. **Type `plotCity` for Project Name.**
5. **Change the Project Location to any directory on your computer and click Next.**
6. **Select the GlassFish V3 TP2 Server for the server.**
7. **Keep the other settings at their default and Click Next.**
8. **Select the jMaki Ajax Framework and select the No CSS layout template.**
9. **Click Finish.**

▼ Adding the Widgets to the Page

With this task, you will add the thisState and thisCity combobox widgets and the Google map widget to the page.

1. **Open `index.jsp` in the editor pane if it is not already there.**
2. **Expand the jMaki Dojo node in the jMaki Palette, located on the right side of the IDE window.**
3. **Select and drag two Combobox widgets onto `index.jsp`**

4 **Save your changes.**

You should now see the following tags on index.jsp:

```
<%@ taglib prefix="a" uri="http://jmaki/v1.0/jsp" %>
...
<a:widget name="dojo.combobox"
   value="[
       {label : 'Alabama', value : 'AL'},
       {label : 'California', value : 'CA'},
       {label : 'New York', value : 'NY', selected : true},
       {label : 'Texas', value : 'TX'}
    ]" />
<a:widget name="dojo.combobox"
   value="[
       {label : 'Alabama', value : 'AL'},
       {label : 'California', value : 'CA'},
       {label : 'New York', value : 'NY', selected : true},
       {label : 'Texas', value : 'TX'}
    ]" />
```

5 **Add the ID,** thisState **to the first Combobox widget and the ID,** thisCity **to the second combobox widget, so that the page now looks like the following:**

```
<%@ taglib prefix="a" uri="http://jmaki/v1.0/jsp" %>
...
<a:widget id="thisState" name="dojo.combobox"
   value="[
       {label : 'Alabama', value : 'AL'},
       {label : 'California', value : 'CA'},
       {label : 'New York', value : 'NY', selected : true},
       {label : 'Texas', value : 'TX'}
    ]" />
<a:widget id="thisCity" name="dojo.combobox"
   value="[
       {label : 'Alabama', value : 'AL'},
       {label : 'California', value : 'CA'},
       {label : 'New York', value : 'NY', selected : true},
       {label : 'Texas', value : 'TX'}
    ]" />
```

6 **Add the text,** Select a state**: right before the first Combobox widget.**

7 **Add a paragraph tag and the text,** Select a city: **right before the second combobox widget.**

The page should now look like this:

```
<%@ taglib prefix="a" uri="http://jmaki/v1.0/jsp" %>
...
Select a state:
<a:widget id="thisState" name="dojo.combobox"
```

```
   value="[
       {label : 'Alabama', value : 'AL'},
       {label : 'California', value : 'CA'},
       {label : 'New York', value : 'NY', selected : true},
       {label : 'Texas', value : 'TX'}
    ]" />
<p>
Select a city:
<a:widget id="thisCity" name="dojo.combobox"
   value="[
       {label : 'Alabama', value : 'AL'},
       {label : 'California', value : 'CA'},
       {label : 'New York', value : 'NY', selected : true},
       {label : 'Texas', value : 'TX'}
    ]" />
```

8 Expand the jMaki Google node in the jMaki Palette, located on the right side of the IDE window.

9 Select and drag a Google Map widget onto `index.jsp`, right after the Combobox widgets.

10 Add another paragraph tag directly before the Google map widget.

The page should now look like this:

```
<%@ taglib prefix="a" uri="http://jmaki/v1.0/jsp" %>
...
Select a state:
<a:widget id="thisState" name="dojo.combobox"
   value="[
       {label : 'Alabama', value : 'AL'},
       {label : 'California', value : 'CA'},
       {label : 'New York', value : 'NY', selected : true},
       {label : 'Texas', value : 'TX'}
    ]" />

<p>
Select a city:
<a:widget id="thisCity" name="dojo.combobox"
   value="[
       {label : 'Alabama', value : 'AL'},
       {label : 'California', value : 'CA'},
       {label : 'New York', value : 'NY', selected : true},
       {label : 'Texas', value : 'TX'}
    ]" />
 <p>
 <a:widget  name="google.map"
    args="{ centerLat : 37.4041960114344,
    centerLon : -122.008194923401 }" />
```

11 **Save** `index.jsp`**.**

12 **(Optional) Run the application by right-clicking the project node and selecting Run.**

You will see the three widgets on the page in the browser, but the combobox widgets are populated with the default set of data, and nothing happens when you select values from them. With the next task, you'll load the appropriate data into the combobox widgets.

Loading Data Into a Widget in the `plotCity` Application

This section details the tasks required to populate a widget with data, using the comboboxes included in the `plotCity` application as examples. These tasks are:

- Creating the Server-Side Data
- Creating the JavaBeans™ Component to Hold the Data
- Accessing the Data From the Page

▼ Creating the Server-Side Data

With this task, you are creating a properties file that contains a set of states and a set of cities for each state. This file acts as the database for the application for simplicity's sake.

1 **Expand the plotCity node and right-click on the Source Packages node.**

2 **Select New > Java Package.**

3 **Enter plotCity as the package name.**

4 **Click Finish.**

5 **Right-click on the project in the Projects window and choose New > Other**

6 **In the New File dialog, select Other from the Categories pane.**

7 **Select Properties File from the File Types pane.**

8 **Click Next.**

9 **Enter** `cities` **in the File Name field.**

10 **Click Finish.**

11 **Select** `plotCity/src/java/plotCity` **as the location of the properties file.**

12 Copy the following content to the `cities.properties` file:

```
AK=Anchorage,Fairbanks,Juneau,Nome
AZ=Mesa,Phoenix,Scottsdale,Tucson
CA=Los Angeles,Sacramento,San Diego,San Francisco
OR=Bend,Eugene,Portland,Salem
WA=Olympia,Seattle,Spokane,Tacoma
```

13 Save the file.

▼ Creating a JavaBeans Component That Holds the Data

By performing this task, you are creating the bean that holds the state and city data that you can use with the comboboxes.

1 Expand the plotCity > Source Packages nodes.

2 Right-click the `plotCity` package and choose New > Java class.

3 Type `StateBean` for the Class Name, `plotCity` for the Package and click Finish.

4 Add the following variable declarations to the class:

```
private String[] states =
    new String [] {"Alaska", "Arizona", "California", "Oregon"};
private String[] stateCodes =
    new String[]{"AK", "AZ", "CA", "OR"};
protected String[] AKCities =
    new String[] {"Anchorage", "Fairbanks", "Juneau", "Nome"};
private ResourceBundle cityNames = null;
```

5 Add an init method after the constructor that creates a `ResourceBundle` object from the `cities.properties` file:

```
private void init() {
 cityNames = ResourceBundle.getBundle("plotCity.cities");
}
```

6 Add a call to the `init` method inside the `StateBean` constructor:

```
public StateBean(){
    this.init();
}
```

7 Add the following `getStates` method to the class. It loads a JSON array with the list of state names that you initialized in step 4:

```
public String getStates() throws JSONException {
    JSONArray statesData = new JSONArray();
    JSONObject stateData = new JSONObject();
```

```
    for (int loop = 0; loop < states.length; loop++) {
        stateData.put("label", states[loop]);
        stateData.put("value", stateCodes[loop]);
        statesData.put(stateData);
        stateData = new JSONObject();
    }
    return jmaki.util.JSONUtil.jsonArrayToString(statesData, new StringBuffer());
}
```

8 Add the following `getCities` method to the class. It loads a JSON array with the list of Alaskan city names that you initialized in step 4:

```
public String getCities() throws Exception {
    JSONObject cityData = new JSONObject();
    JSONArray citiesData = new JSONArray();
    for(int i = 0; i < AKCities.length; i++){
        cityData.put("label", AKCities[i]).toString();
        cityData.put("value", AKCities[i]).toString();
        citiesData.put(cityData);
        cityData = new JSONObject();
    }
    return jmaki.util.JSONUtil.jsonArrayToString(citiesData, new StringBuffer());
}
```

9 Add the following `getNewCities` method to the class. It loads a JSON array with the list of city names for the selected state:

```
public String getNewCities(String state) throws JSONException {
    JSONObject city = new JSONObject();
    JSONArray cities = new JSONArray();
    String[] names = null;
    try {
        names = cityNames.getString(state).split(",");
    } catch(Exception e){
        return null;
    }
    for(int i = 0; i < names.length; i++){
        city.put("label", names[i]);
        city.put("value", names[i]);
        cities.put(city);
        city = new JSONObject();
    }
    return jmaki.util.JSONUtil.jsonArrayToString(cities, new StringBuffer());
}
```

10 Right-click the editor pane and select Fix Imports from the pop-up menu. The IDE will import the packages the class needs.

11 **Save** `StateBean.java`

▼ Accessing Data From the Page

With this task, you will initialize the combobox widgets with the server-side data.

1 **Open** `index.jsp` **into the source editor pane.**

2 **Expand the JSP node in the Palette and select and drag Use Bean into the Source Editor below the** `h1` **tag.**

3 **In the Insert Use Bean dialog box, enter the following values and then click OK.**

- ID: `StateBean`
- Class: `plotCity.StateBean`
- Scope: `session`

The IDE generates the following tag:

```
<jsp:useBean id="StateBean" scope="session"
    class="plotCity.StateBean" />
```

This tag gives `index.jsp` access to the `StateBean` component.

4 **Replace the value attribute of the** `thisState` **tag with an EL expression that points to the** `states` **property of** `StateBean`**:**

```
value="${StateBean.states}"
```

5 **Replace the** `value` **element of the** `thisCity` **tag with an EL expression that points to the** `cities` **property of** `StateBean`**.**

```
value="${StateBean.cities}"
```

6 **Save the file.**

7 **(Optional) Right-click the project node and select Run.**

Your browser should open and show the two comboboxes populated with the data your created. With the next set of tasks, you will add the event-handling mechanism so that you can select a city and plot it on the map.

Handling Events in the `plotCity` Application

This section details how you can use the publish/subscribe mechanism to handle events by performing the following tasks:

- Publishing to a topic

- Writing an Event Handler
- Subscribing to a topic

Publishing to a Topic

After completing this task, thisState will subscribe to the /cb/getState topic.

1 **Open** index.jsp **in a source editor pane.**

2 **Add a** subscribe **attribute to the** thisState **combobox and set it to the topic,** /cb/getState**:**

```
<a:widget  id="thisState"
    name="dojo.combobox"
    publish="/cb/getState"
    value="${StateBean.states}" />
```

3 **Save** index.jsp**.**

Writing a Handler that Handles the Widget Event

After this task is complete, the plotCity application will be able to populate the thisCity combobox widget with new data when the user selects a state from the thisState widget.

1 **Open the** glue.js **file, located in** plotCity/Web Pages**, in the editor pane.**

2 **At the end of the** glue.js **file, add the following** subscribe **function:**

```
jmaki.subscribe("/cb/getState/*", function(args) {
    var message = args.value;
    jmaki.doAjax({method: "POST",
    url: "Service?message=" + encodeURIComponent(message),
    callback: function(_req) {
        var tmp = _req.responseText;
        var obj = eval("(" + tmp + ")");
        jmaki.publish('/cb/setValues', obj);
        // handle any errors
        }
    });
});
```

▼ Writing a Servlet to Obtain the Server-Side Data for the Handler

While performing the previous task, you added an Ajax call to the Service servlet to obtain the values from StateBean. You'll create the servlet with this task.

1 **Expand the plotCity > Source Packages > plotCity node.**

2 **Right-click on the plotCity package icon and select New > Servlet.**

3 **Enter Service for the class name.**

4 **Click Finish.**

5 **Delete everything inside the servlet class so that all that the file contains are the package statement, the import statements automatically generated, and the following empty servlet class:**

```
public class Service extends HTTPServlet{}
```

6 **Add the following private variable declaration:**

```
private ServletContext context;
```

7 **Add the following init method to the class:**

```
@Override
public void init(ServletConfig config)
    throws ServletException {
    this.context = config.getServletContext();
}
```

8 **Add the following doPost method to the class:**

```
public void doPost(
    HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
    HttpSession session = request.getSession();
    StateBean stateBean =
        (StateBean)session.getAttribute("StateBean");
    String cityData = new String();
    String message = request.getParameter("message");
    try{
        cityData = stateBean.getNewCities(message);
    } catch (Exception e){
        System.out.println("could not get city data");
    }
    PrintWriter writer = response.getWriter();
```

```
    writer.write(cityData);
    session.setAttribute("stateBean", stateBean);
}
```

This method gets the message parameter from the request, passes it to `getNewCities`, and passes the resulting list of cities to the response.

9 **Save the file.**

▼ Subscribe to a Topic

With this task, you will get `thisCities` to subscribe to the `/cb/setValues` topic so that it populates with the cities returned from `StateBean`.

1 **Expand the plotCity > Web Pages nodes.**

2 **Open `index.jsp` in a editor pane.**

3 **Add a `subscribe` attribute to the `thisCity` widget and give it the topic `/cb/setValues`, as shown here:**

```
<a:widget id="thisCity"
    name="dojo.combobox"
    subscribe="/cb"
    value="${StateBean.cities}" />
```

4 **Save the file.**

5 **(Optional) Run the example by right-clicking the project node and selecting Run.**

Now you can select a state and see the `thisCity` combobox populate with the names of cities located in that state. With the next task, you'll make it so the application will plot a city on the map when you select the city.

▼ Accessing an External Service from a Widget

1 **Expand the plotCity > Web Pages node.**

2 **Open `index.jsp` in the editor pane.**

3 **Add a `publish` attribute to the `thisCity` combobox widget and set it to `/cities`:**

```
<a:widget id="thisCity"
    name="dojo.combobox"
    publish="/cities"
```

```
    subscribe="/cb"
    value="${StateBean.cities}" />
```

4 **Expand the plotCity > Web Pages node.**

5 **Open** `glue.js` **in a source editor.**

6 **Add the following handler to the end of** `glue.js`**:**

```
jmaki.subscribe("/cities/onSelect", function(item) {
    var city = item.value;
    var state = jmaki.attributes.get('thisState').getValue();
    var location = city + ", " + state;
    var encodedLocation = encodeURIComponent("location=" + location);
    var url = jmaki.xhp +
        "?id=yahoogeocoder&urlparams=" +
        encodedLocation;
    jmaki.doAjax({url: url, callback : function(req) {
        if (req.responseText.length > 0) {
            // convert the response to an object
            var response = eval("(" + req.responseText + ")");
            var coordinates = response.coordinates;
            v = {results:coordinates};
            jmaki.publish("/jmaki/plotmap", coordinates);
        } else {
            jmaki.log("Failed to get coordinates for " +
                location );
        }
    }
});
});
```

7 **Save** `glue.js`**.**

▼ Deploying and Running on the GlassFish v3 Technology Preview 2 Application Server

1 **Right-click on the** `plotCity` **project node.**

2 **Select Properties.**

3 **Select Run.**

4 **Select GlassFish V3 TP2 from the Server combobox.**

5 **Click OK.**

6 **Right-click on the `plotCity` project node.**

7 **Select Run project.**

Index

D
developing the plotCity application, 18

E
event handling, 13
external services, 15

J
jMaki web application, creating a, 8
jMaki widget
  adding a, 9
  initializing with data, 11
  loading data into a, 11
  updating data in a, 12

P
plug-ins
  GlassFish v3 JavaEE Integration, 6
  jMaki Ajax Support, 6
publish/ subscribe system, 13

T
the plotCity example, 7