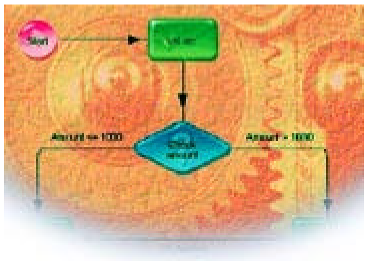
Lotus Developer Domain
The Technical Resource for Lotus software
LDD Today



Customizing Lotus Workflow Web Viewer and enabling it in Java

Level: Intermediate
Works with: Lotus Workflow
Updated: 01-Jul-2003

by Keri Tuttle
and Seol Young Park

An important and useful feature of Lotus Workflow is the Web Viewer. The Web Viewer lets you view a job or process diagram and examine its status, properties, and activities. You can also print a job or process diagram from the Web Viewer.

In this article, we describe how to enable and use the Web Viewer included in our Lotus Workflow Java API demo. This custom Web Viewer can be accessed by Web clients running on WebSphere Application Server. We begin by installing the Web Viewer and configuring it for our demo. We then describe Web Viewer customization features that allow you to make it accessible from a client's Web page. (You can download the demo, complete with all sample code shown in this article, from the Sandbox.)

This article assumes that you're an experienced Java/JSP/JavaScript programmer and that you are familiar with Lotus Workflow features and terminology. We also assume that you're experienced with the Domino and WebSphere environments, especially WebSphere Application Server and WebSphere Studio Application Developer. In addition, you should read the earlier *LDD Today* article Using the Lotus Workflow Java API in Domino and WebSphere, which covered how to use the Lotus Workflow Java API in Domino and WebSphere.

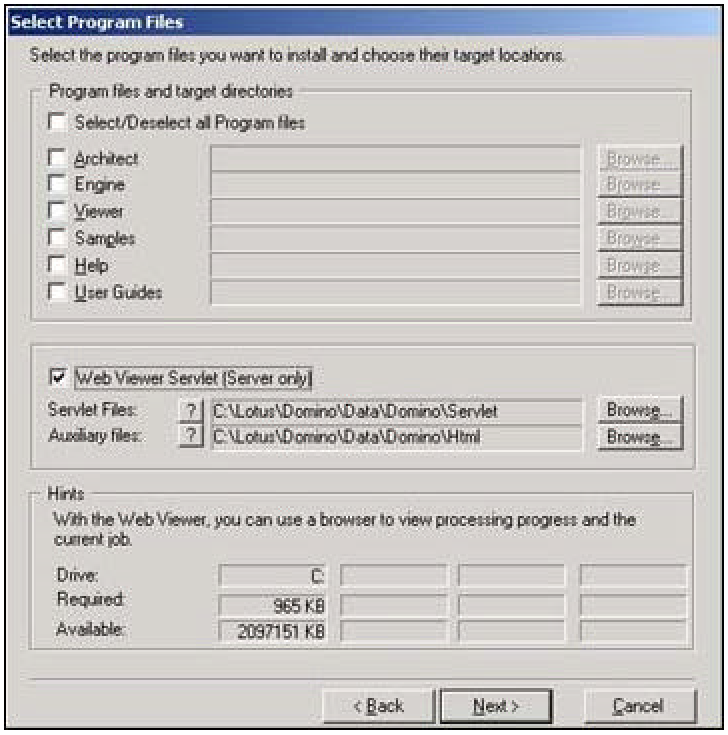
Installing and configuring the Web Viewer

The first thing we need to do is install and configure Lotus Workflow.

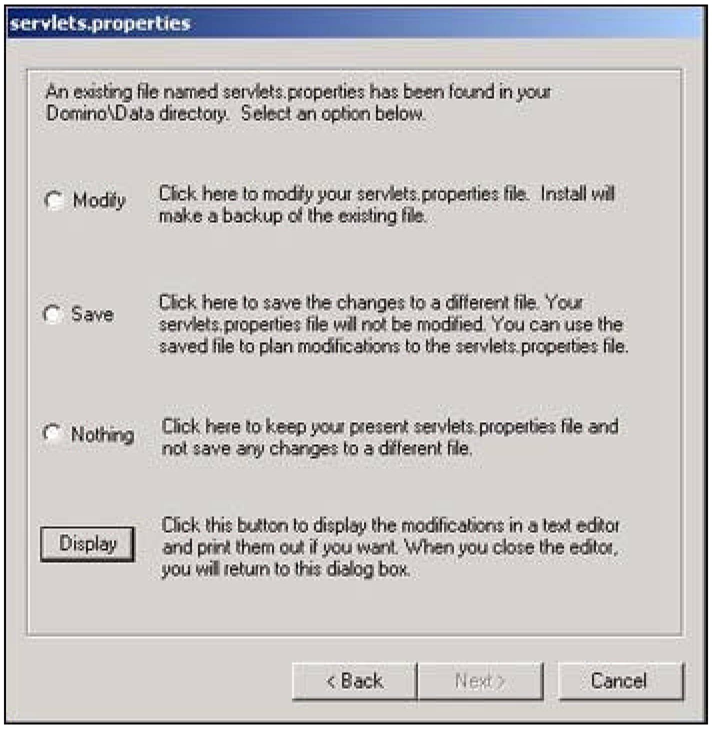
Installation

There are two parts to the Lotus Workflow install: the main install for the Workflow Engine and Architect and the Web Viewer install. This section describes only the Web Viewer portion of the installation. For a full description of the Lotus Workflow installation procedure, see the Lotus Workflow Installation and Administration Guide.

1. Run the setup for Lotus Workflow. When you see the following screen, deselect the Select/Deselect all Program files checkbox, and select the Web Viewer Servlet (Server only) checkbox:


Select Program Files
Select the program files you want to install and choose their target locations.
Program files and target directories
☐ Select/Deselect all Program files
☐ Architect
☐ Engine
☐ Viewer
☐ Samples
☐ Help
☐ User Guides
Browse...
☑ Web Viewer Servlet (Server only)
Servlet Files: C:\Lotus\Domino\Data\Domino\Servlet
Auxiliary files: C:\Lotus\Domino\Data\Domino\Html
Hints
With the Web Viewer, you can use a browser to view processing progress and the current job.
Drive: C:
Required: 965 KB
Available: 2097151 KB
< Back
Next >
Cancel

2. A prompt asks if you want setup to detect the Domino server. Click Yes. The program then enters the proper paths for the Web Viewer.
3. Click Next to start the install.
4. If you do not have a servlets.properties file on your system, the install program asks if you want one created. Click Yes. If you do have a servlets.properties file in your system, you see the following screen. Choose the appropriate option and click Next.


servlets.properties

An existing file named servlets.properties has been found in your Domino\Data directory. Select an option below.

○ Modify — Click here to modify your servlets.properties file. Install will make a backup of the existing file.

○ Save — Click here to save the changes to a different file. Your servlets.properties file will not be modified. You can use the saved file to plan modifications to the servlets.properties file.

○ Nothing — Click here to keep your present servlets.properties file and not save any changes to a different file.

Display — Click this button to display the modifications in a text editor and print them out if you want. When you close the editor, you will return to this dialog box.

The Web Viewer files are now copied to your hard drive. Accept the defaults to finish the rest of the installation.

Configuration
There are a few settings in Domino that need to be set for the Web Viewer to work. The Domino servlet engine is very important for the functioning of the Web Viewer. If it is not enabled, the Web Viewer cannot run. The Domino server also needs access to run Java. Below are the steps to enable Java support and to grant the required access rights. For more information on Domino Java support, see the **Domino Administrator Help**.

Enabling servlet support
Follow these steps to enable Domino servlet support.
1. In the Domino Directory, open the Server document for your server.
2. Open the Internet protocols - Domino Web Engine tab.
3. In the Java servlet support field, select Domino Servlet Engine.
4. In the Servlet URL field, enter /servlet (if it is not there already).
5. In the Class path field, enter domino\servlet (if not there already).

Allowing Java to run on the server
Follow these steps to enable Domino to run Java applications.
1. Open the Server document for your server.
2. Click the Security tab.
3. Enter an asterisk (*) in the two fields labeled Run restricted Java/JavaScript/COM and Run unrestricted Java/JavaScript/COM.
4. Save and close the document.

The configuration information provided in the preceding steps is for the demo provided with this article. To use your own databases, you need to add your application database path to the servlet.properties file and enable the viewer log in the application database setup document. Also, the application database's ACL must give the server Manager access with the files Process Server and Process Reader. For more information on configuring your own application database, see the Lotus Workflow documentation.

Customizing the Web Viewer
There are a number of ways you can change the "look and feel" of the Web Viewer so that it blends seamlessly with a client's Web page. You can select different style sheets or choose to have Sametime awareness enabled for instant messaging. This section describes some of the modifications you can make. (Note that the modifications presented here can also be used in a pure Domino environment.)

The following table shows some of the parameters you can add to your servlets.properties file to change the look and feel of the Web Viewer:

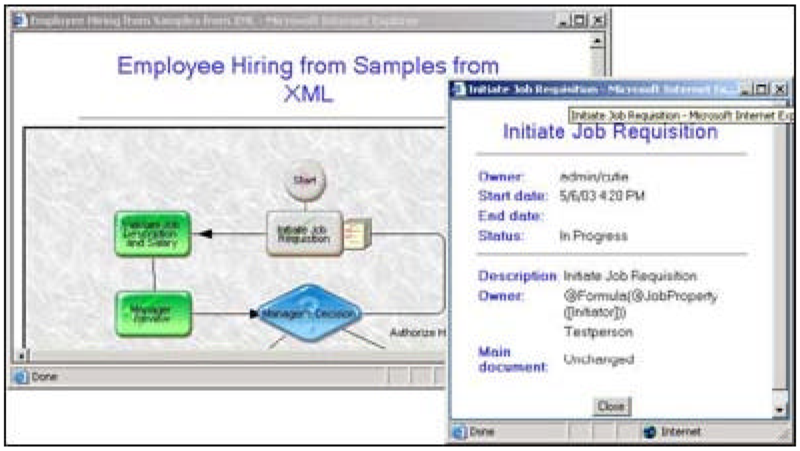
Attribute	Values	Description
ApplPath	<Path to application database file>	Specifies the path to the application database file. The default is "" (no path specified).
LogLevel	debug, status, error, none	These are as follows: • debug: all messages logged • status: all status and error messages logged • error: only error messages logged • none: logging is disabled If omitted, default is status.
Style	standard, glass, sketch, white	Specifies the style sheet to use when drawing diagrams. If omitted, default is glass.
DiagramTitle	true, false	If false, turns off diagram page titles. If omitted, default is true.
PropertyTitle	true, false	If false, turns off property page titles. If omitted,

		default is true.
CloseButton	true, false	If false, turns off the close button on property pages. If omitted, default is true.
Sametime	true, false	If false, turns off Sametime on property pages. If omitted, default is true.
SametimeServer	<Server name>	Specifies the name of the Sametime server. If omitted, Sametime is turned off.
NoLogin	true, false	If true, then no SSO cookie is sought and no user ID/password pair is requested. Instead, all access is through the server's Notes ID (usually permitting universal access to the Lotus Workflow databases).
LoginPage	<Login URL>	Specifies the URL used instead of the servlet's default page, if a user ID/password pair is requested. Note that this specifies the scheme (for example, http or https) as well as address.
LogoutPage	<Logout page URL>	Specifies the page that is returned if a request type of logout is received. If null, a blank page is sent. The default value is null.

Here is an example of a servlet.properties file edited to add Sametime support, to remove the diagram title, to remove the close buttons, and to change the style from glass to sketch. Also note there is no title for your diagram, and the style sheet has changed. If you click an activity, you also find that the close button is no longer available:

```
servlet.WebViewer.initArgs=LogLevel=DEBUG,ServerAlias=WebViewer,ApplPath=
Workflow/Sample_Application_301_en.nsf,Sametime=true,SametimeServer=server.lotus.com,
DiagramTitle=False,CloseButton=False,Style=sketch
```

The following graphics compare the default settings for the Web Viewer and the settings added in the preceding example. First let's look at the default:

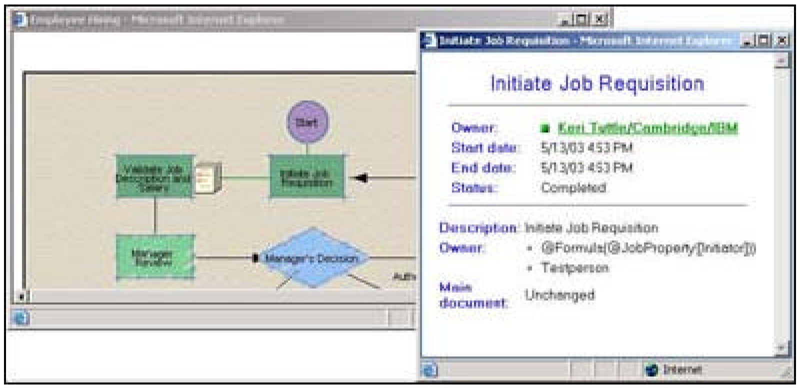


Initiate Job Requisition

Owner: admin/cutie
Start date: 5/6/03 4:20 PM
End date:
Status: In Progress

Description: Initiate Job Requisition
Owner: @Formula(@JobProperty([Initiator]))
Testperson
Main document: Unchanged

And here's the same view customized:



Initiate Job Requisition

Owner: Keri Tuttle/Cambridge/IBM
Start date: 5/13/03 4:53 PM
End date: 5/13/03 4:53 PM
Status: Completed

Description: Initiate Job Requisition
Owner:
- @Formula(@JobProperty([Initiator]))
- Testperson

Main document: Unchanged

Adding another application database for viewing

A great feature for sites that have more than one application database is the multiple Web Viewer servlet. This allows the Web Viewer to work with more than one application database at a time. Here is how to add multiple application support to the Web Viewer:

1. Open your servlets.properties file.
2. Add these two lines to the file:

   ```
   servlet.WebViewer2.code=com.lotus.wf.WFServlet
   servlet.WebViewer2.initArgs=LogLevel=DEBUG,ServerAlias=WebViewer2,
   ApplPath=workflow/app2.nsf
   ```

   replacing app.nsf for your other application database.

3. Open your application specified in step 2.
4. Choose View - Design.
5. Click Resources, then click Shared Fields.
6. Open SKWebViewerSettingsOS and with Value selected on the Objects tab, enter the following:

   ```
   FIELD WebViewerURLOS:="/servlet/WebViewer2";
   ```

7. Restart the HTTP task on the Domino server.

Now you can use the Web Viewer with both our demo application database and your own application database. You can use this setting for as many application databases as you want.

Sametime settings (optional)

To integrate Sametime functionality into your Web Viewer:

1. Copy the Sametime folder from the Workflow install CD to the following directory on your Sametime server:

   ```
   <drive>\Lotus\Domino\Data\domino\html\stlinks
   ```

2. In the servlets.properties file, add the following to your Web Viewer line:

   ```
   Sametime=true, SametimeServer=Sametime4.yourcompany.com
   ```

   The line should now look like this:

   ```
   servlet.WebViewer.initArgs=ServerAlias=WebViewer,ApplPath=
   Workflow/<Application database filename>,Sametime=true,
   SametimeServer=Sametime4.yourcompany.com
   ```

3. Open the Domino Notes.ini file and add the following to the JavaUsersClasses line:

   ```
   <jar file path here>\STComm20.jar
   ```

Customizing the style sheet WebViewer.css and JavaScript files

There are two additional areas where customization can be done on the Web Viewer. These are the style sheet WebViewer.css file and the JavaScript file.

WebViewer.css
Style settings are not embedded directly in the HTML but are controlled through a separate Cascading Style Sheet file called WebViewer.css. By modifying this file, you can control the appearance of the page by adjusting title fonts, font colors, font sizes, and the background color.

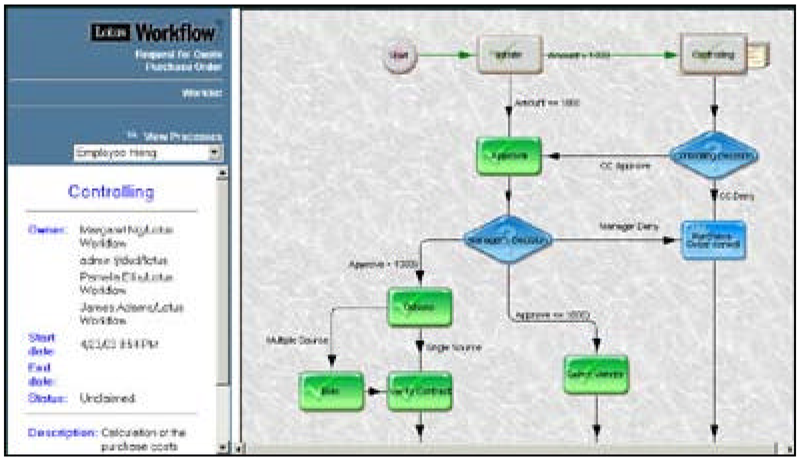
JavaScript
Servlet requests are usually routed through JavaScript functions in the Web Viewer so that you can easily use them for customization. For example, when a user clicks on an activity in a View Diagram, the Property Page is not loaded immediately. Instead, the JavaScript function onObjectClick is called to bring up the correct Property Page. By using JavaScript as an intermediary between pages, you have a layer of functionality that can be modified.

Currently only three JavaScript functions are available for the Web Viewer:

- displayDiagram makes a request to the WebViewer Servlet to display a Process or Process Instance Diagram.
- displayProperties makes a request to the WebViewer Servlet to display a Property Page that corresponds to the object selected.
- onObjectClick is called when an object has been clicked in the View Diagram. For most objects, onObjectClick calls displayProperties to display a Property Page that corresponds to the object selected. However, for clusters or process links, it calls displayDiagram to expand the cluster or sub-process.

Enabling the Web Viewer

In this section, we enhance our Java API demo to show all the available process names for the user to start a workflow with. We also show the current design of the selected process or the current status of a selected job from the worklist view in various customized ways. For example, the following screen shows the status of the selected job in a workflow process:



We display detailed information on the left bottom when the user clicks the controlling activity. You also can open diagrams and property pages in new windows based on your design.

We need to customize code in both your WebSphere and Domino environments to accomplish this. We discuss the changes in Domino first and then move to the Java API changes in WebSphere. (Our demo provides all the customized files for both Domino and WebSphere. You can download our demo from the **Sandbox**.)

Customizing Domino

We need to modify the servlet initialization arguments, StyleSheet file, and JavaScript file. Before you begin, we suggest you back up your files.

Step 1: Modify servlet.properties on your Domino server
Edit the servlet.properties in your Domino server as indicated by the bold text. You can find this file in your Domino data directory. We've added two servlets initialization arguments to turn off diagram title and close button by default:

```
servlet.WebViewer.initArgs=LogLevel=DEBUG,ServerAlias=WebViewer,ApplPath=
Workflow/Sample_Application_301_en.nsf, DiagramTitle=false,CloseButton=false
```

Step 2: Modify the WebViewer.css file in your Domino data\domino\html\WfRoot\StyleSheets directory
The following lines add the border line effect to the property page style so that it shows with the border line in a frame or a new window:

```
body.propbody
    { ................
    border : 2px inset solid #517999;
    ............
    }
```

Step 3: Modify the WebViewer.js file under Domino data\domino\html\WfRoot\Scripts
If you customize only the process/job diagram, you can use a separate file to store your custom codes. However, the custom file is not accessible in the diagram pages submitted by Web Viewer, and therefore may not work as expected. To avoid this, we've added custom JavaScript code inside the original file with separate code using a new function object called CustomViewer.

The JavaScript object CustomViewer is created when the page is loaded by the line at the end in WebViewer.js file:

```
var myViewer = new CustomViewer();
```

You can reference CustomViewer properties and methods by calling it as myViewer, for instance, myViewer.windowFeature. CustomViewer has several properties such as target frame name, window features (when opening as a new window), Web Viewer constants such as diagram styles, and diagram types as follows:

```
function CustomViewer() {
    this.isOpenNewWindow = true;
    this.diagramTargetFrame = "";
    this.propertyTargetFrame = "leftBottomF";
    this.windowFeature =
    "width=600,height=400,menubar=no,toolbar=no,location=no,status=yes,resizable=yes,
    scrollbars=yes";
};

// properties for WebViewer constants
CustomViewer.prototype.styles = [ 'glass','sketch','standard','white']; // We will pass 0 ~ 3 to
set the diagram style with the corresponding name
CustomViewer.prototype.processDiagram =  'Process';
CustomViewer.prototype.jobDiagram =  'Job';
```

The custom object also has a utility method to get the current selected process name from the available process list and two wrap-up methods to call Web Viewer functions to display diagrams:

- *showProcess* displays the process diagram
- *showJob* displays the selected job's diagram

```
// A method to pass necessary parameters to display a process diagram.
CustomViewer.prototype.showProcess = function ( fieldName, styleId, isNewWindow)
{
    var obj = eval('document.forms[0].'+fieldName);
    // Note that there is a known issue in the current API to return a proper process ID, so we are manipulating
    the process ID here to work around it.
    processID = this.getSelectedProcess(obj)+"A0";
    // If we open the diagram in the existing window, we pass a target frame name.
    if ( ! isNewWindow) {
    this.isOpenNewWindow = false;
    this.diagramTargetFrame = "contentF"; // target frame name that you'd like to open your page.
}
    // It calls the function provided by Lotus Workflow Web Viewer
    displayDiagram(WEB_VIEWER_URL,'',myViewer.processDiagram, processID,'',this.styles[styleId],'');
}

    // A method to pass necessary parameters to display a process diagram.
    // It works almost the same as showProcess shown above except for the parameter list that we pass.
    CustomViewer.prototype.showJob = function (styleId, instanceID, isNewWindow) {
    .........
    displayDiagram(WEB_VIEWER_URL,'',myViewer.jobDiagram, instanceID,'',this.styles[styleId],'');
}
```

For the showJob method, you can use the custom parameter (the last parameter in the list) to enable Sametime. Here is an example:

```
displayDiagram(WEB_VIEWER_URL, '' ,myViewer.jobDiagram, InstanceID, '' , this.styles[styleId] ,
'&Sametime=true&SametimeServer=yourSametimeServer.com');
```

So far, we've explained the new code we added. Now, we slightly modify the two default functions provided by the Lotus Workflow Web Viewer as follows. The rest of the default functionality stays the same:

```
function displayDiagram(servlet, format, diagtyp, diagid, clusid, style, custom)
   {
      ........................................
      // If we set it to open in a new window, it will open the diagram to a new window using the
      window features we set.
      // Otherwise, it will open to a frame in the existing window passed in myViewer.diagramTargetFrame.
      if(myViewer.isOpenNewWindow)
      {
         var window = top.window.open(page, "",  myViewer.windowFeature);
      } else {
         eval("parent." + myViewer.diagramTargetFrame + ".location = page;");
      }
   }

   function displayProperties(servlet, format, diagtyp, diagid, objtyp, objid, custom)
   {
      .................................................

      if(myViewer.propertyTargetFrame=="") {
         propWin = window.open(page, "propertyWindow",  myViewer.windowFeature);
      } else {
         //After Web Viewer submits a diagram, our custom object myViewer will not be available in the
         diagram page.
         //To share the code for both cases opening to new or old window, we have tried to put a trick here.
         //If there is no target frame object, we will open to a new window.
```

```
                if (typeof(parent.leftBottomF)=="object") {
                    eval("parent." + myViewer.propertyTargetFrame + ".location = page;");
                } else {
                    propWin = window.open(page, "propertyWindow",  myViewer.windowFeature);
                }
            }
            ........................
        }
```

This completes our customization in Domino. Now, let's move on to WebSphere.

Customizing WebSphere
Before you begin working with the code, you must import the demo file LWFJavaAPIDemo2.ear into WebSphere Studio Application Developer:

1. Open WebSphere Studio Application Developer.
2. Choose File - Import.
3. Select the EAR file from the displayed list of file types, then click Next.
4. Browse to LWFJavaAPIDemo2.ear.
5. Specify your Enterprise Application project name, and then click Finish.

Note that our demo EAR file has been exported from WebSphere Studio Application Developer 5. If you are using WebSphere Studio Application Developer 4, you need to correct duplicated source directories. Move all the files under source\source to the source directory and remove the nested source directory.

Modify constants files
In addition to the two existing constants (Domino server name and replica ID of the Lotus Workflow application database), we have two new constants that you need to modify for your demo environment. One is for the Web Viewer servlet URL, the other is the location of the Web Viewer JavaScript file:

```
public final class CustomConstants {
        .......
        public final static java.lang.String LOTUS_WEBVIEWER_URL = "
        http://yourDominoServer.com/servlet/WebViewer";
        public final static java.lang.String LOTUS_WEBVIEWER_JSFILE = "
        http://yourDominoServer.com/WFRoot/Scripts/WebViewer.js";

    }
```

We've also moved the link paths and SQL queries used in our pages to the constants class file. You don't need to modify this unless you want to try different links/queries. We used the SQL QUERY_AVAILABLE_PROCESSES to get all the available process names for the user:

```
public final class Constants {
        ............................................
        // SQL queries used in the API calls
        public final static java.lang.String QUERY_AVAILABLE_PROCESSES = "SELECT PROCESSMANAGER,
        NAME FROM AVAILABLE";
         ............................................
    }
```

After modifying these lines, save the file and rebuild the project.

navigator.jsp
Our Java API demo has three frames. One is for navigation on the top left, the second for actual content displayed on the right, and the third is located in the bottom left for the additional information to be displayed (such as property page for each activity in process/job diagram). The navigate.jsp page is shown on the top left to display links to create new jobs, available processes, and the icon to launch process/job diagrams.

The Lotus Workflow Java API getProcessManagers returns the list of processes that can be initiated by the user. We parse the string list and get the process name and save them to another Vector, pName:

```
<%
        Vector availableProcesses = new Vector();
        Vector pName = new Vector();
        try {
            ..........
            availableProcesses =
            worklist.getProcessManagers(Constants.QUERY_AVAILABLE_PROCESSES,-1);

%>
```

We have included the Web Viewer JavaScript file to get the Web Viewer URL from the custom constant that we set in the class:

```
<script type="text/javascript" src="<%= CustomConstants.LOTUS_WEBVIEWER_JSFILE %>"></script>
<script type="text/javascript">
      var WEB_VIEWER_URL = "<%= CustomConstants.LOTUS_WEBVIEWER_URL %>" ;
</script>
```

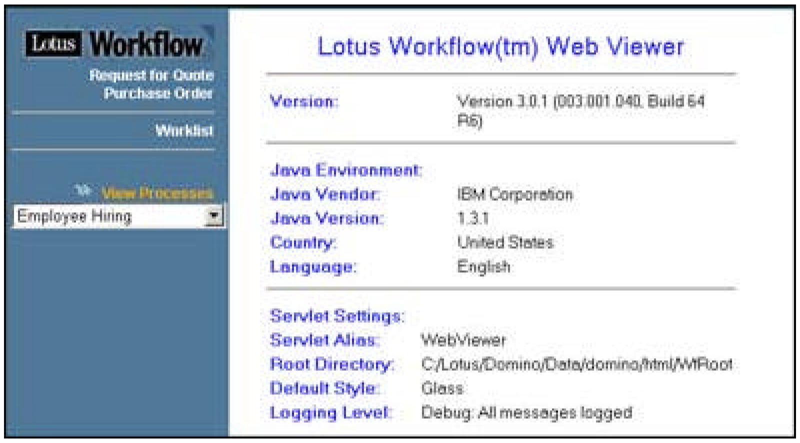
The following code adds a glass icon with the onClick event to display a process diagram with the glass style to an existing frame specified:

```
<IMG class="s-cursorhand" src="../images/lwfvwicnviewpro.gif" border="0"
onClick="myViewer.showProcess('ProcessList',0,false);">
    &nbsp;

        <A href="<%= CustomConstants.LOTUS_WEBVIEWER_URL %>" onClick="myViewer.clean();"
        target="contentF">View
        Processes</A>

        <!-- We parse the strings from availableProcesses object and get the process names to generate the list
        -->
        <SELECT NAME="ProcessList" width="100px">
        <%
            for (int i=0; i < availableProcesses.size() ; i ++) {
            String token = availableProcesses.elementAt(i).toString();
            pName.add(token.substring(token.lastIndexOf("|")+1,token.length()));
        %>
        <OPTION><%= pName.get(i)%>
        </SELECT>
```

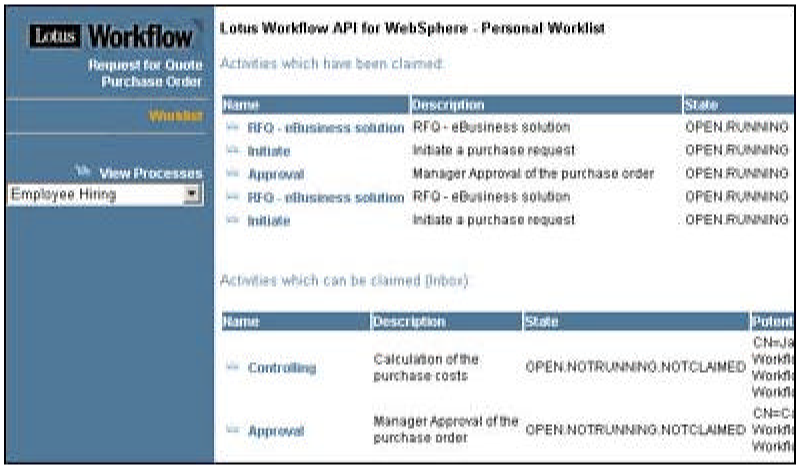
When you click the View Processes text, it opens the Web Viewer servlet information page. You can use this link to test whether or not your Web Viewer is configured and working:



Lotus Workflow(tm) Web Viewer

Version:	Version 3.0.1 (003.001.040, Build 64 R6)
Java Environment:	
Java Vendor:	IBM Corporation
Java Version:	1.3.1
Country:	United States
Language:	English
Servlet Settings:	
Servlet Alias:	WebViewer
Root Directory:	C:/Lotus/Domino/Data/domino/html/WfRoot
Default Style:	Glass
Logging Level:	Debug: All messages logged

ActivityList.jsp
When you click Worklist, notice that we have added an icon in front of each activity. The icon launches the Web Viewer to display each activity status. To demonstrate various ways to display the Web Viewer, we open the Web Viewer in a new window for the activities which have been claimed. The job diagram for new activities is displayed in the existing window:



Lotus Workflow API for WebSphere - Personal Worklist

Activities which have been claimed

Name	Description	State
RFQ - eBusiness solution	RFQ - eBusiness solution	OPEN.RUNNING
Initiate	Initiate a purchase request	OPEN.RUNNING
Approval	Manager Approval of the purchase order	OPEN.RUNNING
RFQ - eBusiness solution	RFQ - eBusiness solution	OPEN.RUNNING
Initiate	Initiate a purchase request	OPEN.RUNNING

Activities which can be claimed (Inbox)

Name	Description	State	Potent
Controlling	Calculation of the purchase costs	OPEN.NOTRUNNING.NOTCLAIMED	CN=Ja Workfl Workfl Workfl
Approval	Manager Approval of the purchase order	OPEN.NOTRUNNING.NOTCLAIMED	CN=C Workfl Workfl

We have added lines to get the instance ID from the key to pass to the Web Viewer when we generate tokens. Tokens[0] has the key returned by the API, and the key string format is:

```
<KeyType>"#"<Server>"#"<ReplicaID>"#"<PhysicalID>"#"<InstanceID>"#"<ActivityDefinitionID>"#"<LoopID>
```

We need the fifth token from the key string:

```
<%
        ......

        String key = java.net.URLDecoder.decode(Tokens[0]);
        for (int idx = 0; idx <4 ; idx++) {
            key =key.substring(key.indexOf("#")+1 , key.length());
        }
        key = key.substring(0, key.indexOf("#"));
%>
```

After we get the key, it's simple to call myViewer.showJob with the glass style diagram to obtain the key and the last parameter to open in a new window or in an existing window:

```
<IMG class="s-cursorhand" src="<%= request.getContextPath() + "/images/lwfvwicnviewpro.gif"
%>"onClick="myViewer.showJob(0,'<%= key %>',true);">
```

Customizing the Web Viewer in your applications

We've shown how you can add Lotus Workflow Web Viewer to our Lotus Workflow Java API demo UI and to the Java API in WebSphere and Domino. Try it out, but don't forget to get authenticated with your Domino server before you run your application in WebSphere Studio Application Developer to obtain an SSO token. We hope you find this useful as a way to customize your own applications.

ABOUT THE AUTHORS
Keri Tuttle joined Lotus/IBM in 2000, originally as a member of the Lotus Workflow quality assurance team. She is now a member of the Lotus Workplace quality assurance team.

Seol Young Park joined Lotus/IBM in 1994, working on multiple projects for the International Product Development team. Seol Young joined the Lotus Workflow team in 1999, serving as a developer for the Lotus Workflow Engine. She is now a member of the Lotus Workplace Messaging Client development team.