ADF Code Corner

71. How-to integrate Java Applets with Oracle ADF Faces

Abstract:

Oracle ADF Faces contains a JavaScript client framework that developers can use to integrate 3rd party technologies like Java Applets and jQuery. This article explains how to establish bi-directional communication between a Java Applet and an ADF Faces application.

twitter.com/adfcodecorner

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu

08-FEB-2011

Introduction

ADF Faces provides developers with a comprehensive list of JavaServer Faces user interface components and behavior tags for building visually attractive web applications. Ideally, to ensure a consistent request lifecycle, developers build web applications with ADF Faces and / or 3rd party JavaServer Faces components alone. However, there may be a need for ADF Faces to integrate with 3rd party technology like Java Applet, JQuery or Flash, when the business case requires it. Integration with 3rd party technology is on the client side with JavaScript, which also means that the application lifecylce and transaction is not shared. In this how-to article, I explain the integration of Oracle ADF Faces with Java Applets for bi-directional communication, as well as the Java Applet calling a managed bean.

In the example, the Java Applet is added to the ADF Faces page as shown below:

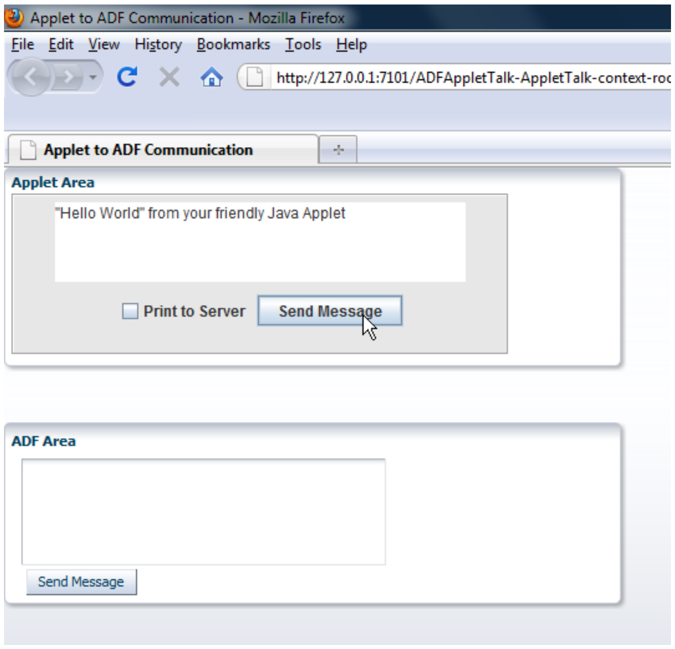
```
 <APPLET code="adf.sample.applet.MessageApplet.class"
                archive="/ADFAppletTalk-AppletTalk-context-root/messageApplet.jar"
                height="130"
                width="400" align="bottom">
                 This browser does not support Applets.
 </APPLET>
```

Note that the APPLET tag is used instead of the EMBED or OBJECT tag that would be needed to explicitly invoke Java run by the Java PlugIn. As shown later, browsers can be configured to use the Java PlugIn instead of its own integrated Java Runtime Rnvironment (JRE), which in many cases does not exist, or is not sufficient to run latest Java.

When installing a client side Java Runtime Environment the browser configuration is added automatically, so that using the APPLET tag is an option okay to use.

Client side only communication

In the example, a Java Applet input text field and a command button are used to send text messages to the ADF Faces input form. The Java Applet uses the `netscape.javascript.JSObject` class contained in jre6\lib\plugin.jar to issue JavaScript calls to the containing page.


Applet to ADF Communication - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:7101/ADFAppletTalk-AppletTalk-context-roo
Applet to ADF Communication
Applet Area
"Hello World" from your friendly Java Applet
Print to Server
Send Message
ADF Area
Send Message

```
private void sendMessageButton_keyPressed(KeyEvent e) {
  int keyCode = e.getKeyCode();
    if (keyCode == 10){
       String message = messageField.getText();
        JSObject win = (JSObject) JSObject.getWindow(this);
        //call the JavaScript messae
        if(printToServer.isSelected()){
          //make sure message is printed to JSF managed bean
          win.eval("callAdfFromJavaMessage('"+message+"',true);");
        }
        else{
          win.eval("callAdfFromJavaMessage('"+message+"',false);");
        }
      }
    }
```

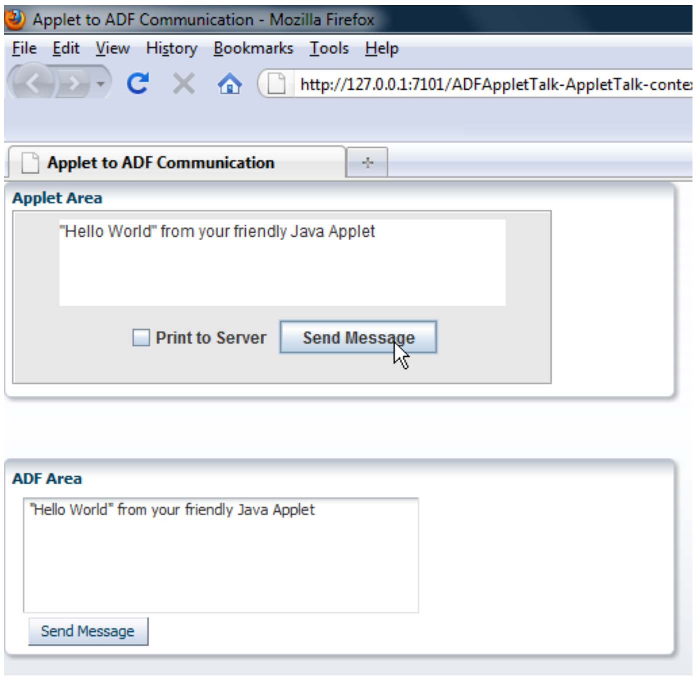
On the ADF Faces page, the JavaScript function `callAdfFromJavaMessage` is defined as shown below:

```
function callAdfFromJavaMessage(messageFromApplet, printToServer){
   var textField = AdfPage.PAGE.findComponentByAbsoluteId("it1");
   textField.setValue(messageFromApplet);
   textField.focus();
      if(printToServer == true){
     //send message as immediate
     AdfCustomEvent.queue(textField,"AppletIsCallingADF",
                          {appletMessage:messageFromApplet},true);
     }
}
```

The JavaScript function uses the `AdfPage.PAGE` API to lookup the ADF Faces input text field component. It then determines the value of the *printToServer* argument to determine whether to send the message to the server too.

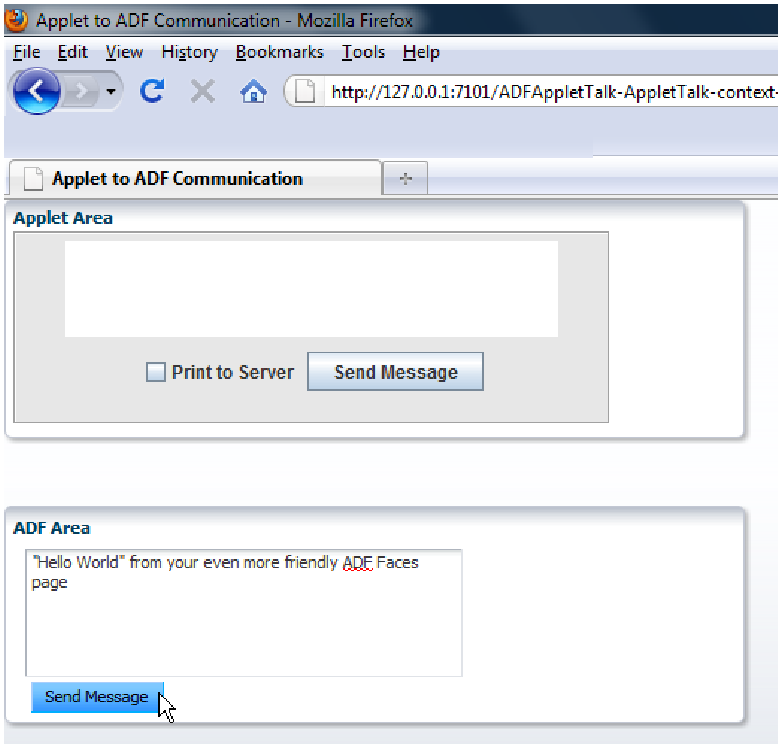
Note: When used in production, all of the JavaScript sources used in this sample should be stored in an external JavaScript library for better performance and code organization.

When a message is sent from the Java Applet, it is printed to the ADF Faces input text field as shown here:


Applet to ADF Communication - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:7101/ADFAppletTalk-AppletTalk-contex
Applet to ADF Communication
Applet Area
"Hello World" from your friendly Java Applet
Print to Server
Send Message
ADF Area
"Hello World" from your friendly Java Applet
Send Message

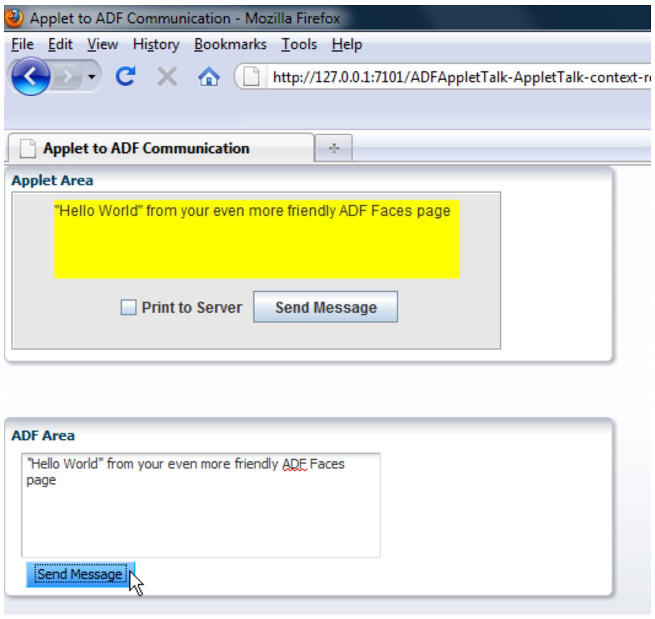
Note that the message is added to the input text field such that submitting the ADF Faces form also submits the message to the Java Server Faces model or ADF binding. Calling a managed bean from a Java

Applet directly therefore is only required for events that should not update input fields on an ADF Faces page but only notify the server.


Applet to ADF Communication - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:7101/ADFAppletTalk-AppletTalk-context-
Applet to ADF Communication
Applet Area
Print to Server
Send Message
ADF Area
Send Message

"Hello World" from your even more friendly ADF Faces page

Public methods of the Java Applet are accessible from JavaScript. In the image below, the ADF Faces form sends a message back to the Java Applet. The Applet displays the message and paints the text field background in yellow.


Applet to ADF Communication - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:7101/ADFAppletTalk-AppletTalk-context-r
Applet to ADF Communication
Applet Area
Print to Server
Send Message
ADF Area
Send Message

"Hello World" from your even more friendly ADF Faces page

"Hello World" from your even more friendly ADF Faces page

The public Java Applet method called from JavaScript is shown here:

```
public void handleJavaScriptToAppletMessage(String message){
  messageField.setBackground(Color.yellow);
  messageField.setText(message);
}
```

The JavaScript function to call into the Java Applet is invoked from an `af:clientListener` tage on the *Send Message* command button.

```
//JavaScript function calling client side
function writeAdfFacesMessageToApplet(evt) {
  var textField = AdfPage.PAGE.findComponentByAbsoluteId("it1");
  var message = textField.getSubmittedValue();
  //talk to Applet
  document.applets[0].handleJavaScriptToAppletMessage(message);
}
```

The Java Applet is accessed by its index in the browser page's `document` JavaScript reference. The JavaScript code reads the current value of the ADF Faces input text field, again using the `AdfPage.PAGE` API to look it up.

Note ADF Faces components must have their *clientComponent* property set to **true** or an `af:clientListener` attached to be accessible from JavaScript. The latter is used in this example.

Java Applet to managed bean communication

A custom event is used in ADF Faces to call a managed bean from a Java Applet. The Java Applet calls a JavaScript function as before, but passes *true* as the value of the *printToServer* argument. The following line of JavaScript queues the incoming Java Applet message as a custom ADF Faces event for a managed bean method to handle:

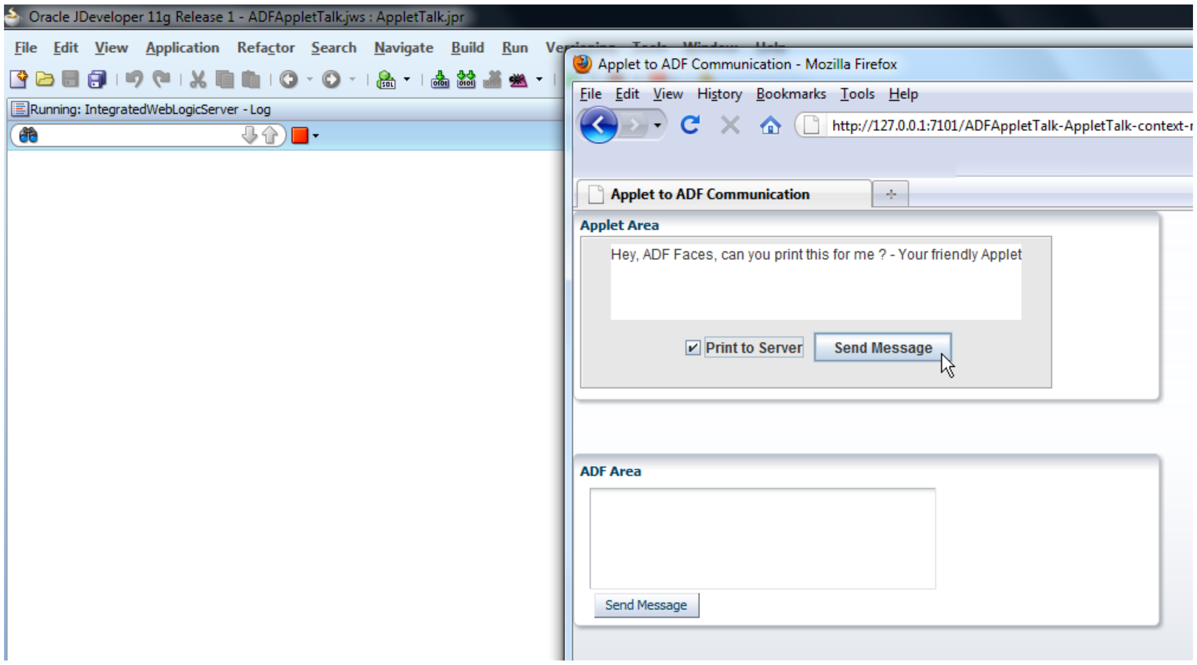
```
AdfCustomEvent.queue(textField,"AppletIsCallingADF",
                     {appletMessage:messageFromApplet},true);
```

The *textField* reference in the above code references the ADF Faces component that has the `af:serverListener` tag defined to call the managed bean method.

The name of the `af.serverListener` tag on the page is *AppletIsCallingADF*.

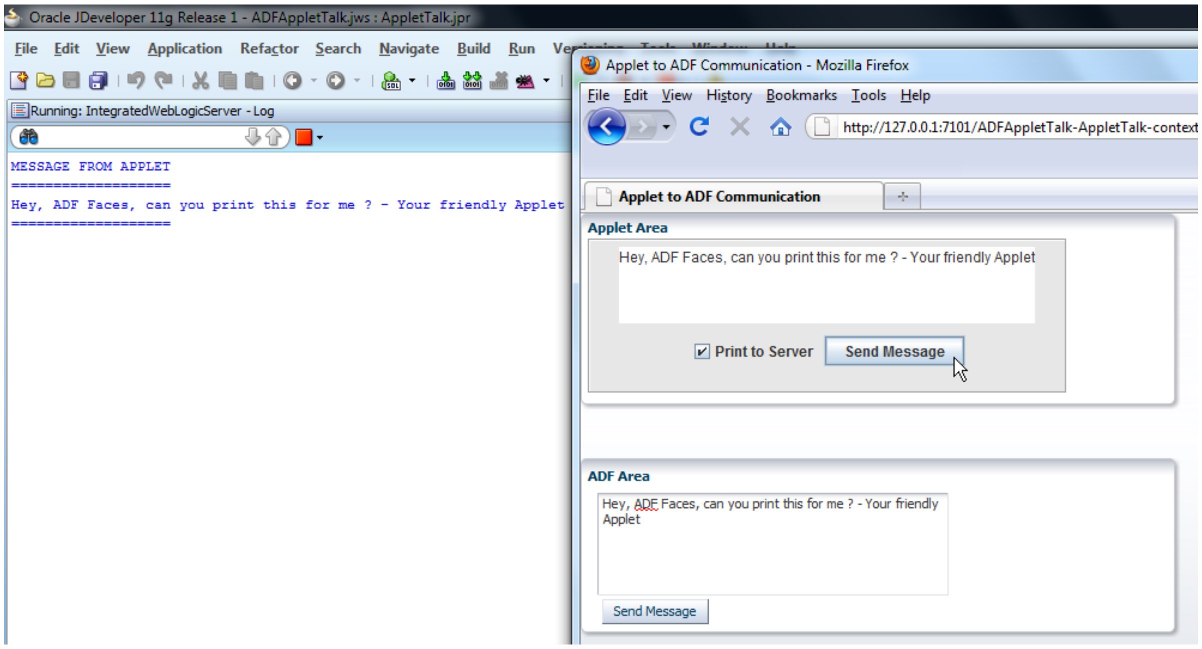
The Java Applet message is passed as the payload, a set of key-value pairs, of the custom event. In the example, a single argument is passed with the name of *appletMessage.*

The last argument specifies how the queued event should be handled by the ADF Faces request lifecycle. In the above example, the argument is set to *immediate* so the server call can be made without client validation to be performed on the ADF Faces page.


Oracle JDeveloper 11g Release 1 - ADFAppletTalk.jws : AppletTalk.jpr
File Edit View Application Refactor Search Navigate Build Run
Running: IntegratedWebLogicServer - Log
Applet to ADF Communication - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:7101/ADFAppletTalk-AppletTalk-context-r
Applet to ADF Communication
Applet Area
Print to Server
Send Message
ADF Area
Send Message

Hey, ADF Faces, can you print this for me ? - Your friendly Applet

As shown in the image above, the Java Applet in this sample uses a checkbox component in Swing for the user to indicate the call to be sent to a managed bean on the server too.


Oracle JDeveloper 11g Release 1 - ADFAppletTalk.jws : AppletTalk.jpr
File Edit View Application Refactor Search Navigate Build Run
Running: IntegratedWebLogicServer - Log
Applet to ADF Communication - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:7101/ADFAppletTalk-AppletTalk-context
Applet to ADF Communication
Applet Area
Print to Server
Send Message
ADF Area
Send Message

```
MESSAGE FROM APPLET
===================
Hey, ADF Faces, can you print this for me ? - Your friendly Applet
===================
```

Hey, ADF Faces, can you print this for me ? - Your friendly Applet

Hey, ADF Faces, can you print this for me ? - Your friendly Applet

The server side managed bean receives an event object of type `ClientEvent` that contains the message sent by the Java Applet.

```
public void onAppletCall(ClientEvent clientEvent) {
        String message = (String)clientEvent.getParameters().get("appletMessage");
        //just print it out
        System.out.println("MESSAGE FROM APPLET");
        System.out.println("===================");
        System.out.println(message);
        System.out.println("===================");
        …
 }
```

For this to work, the `af:serverListener` tag is configured on the input text field as shown below

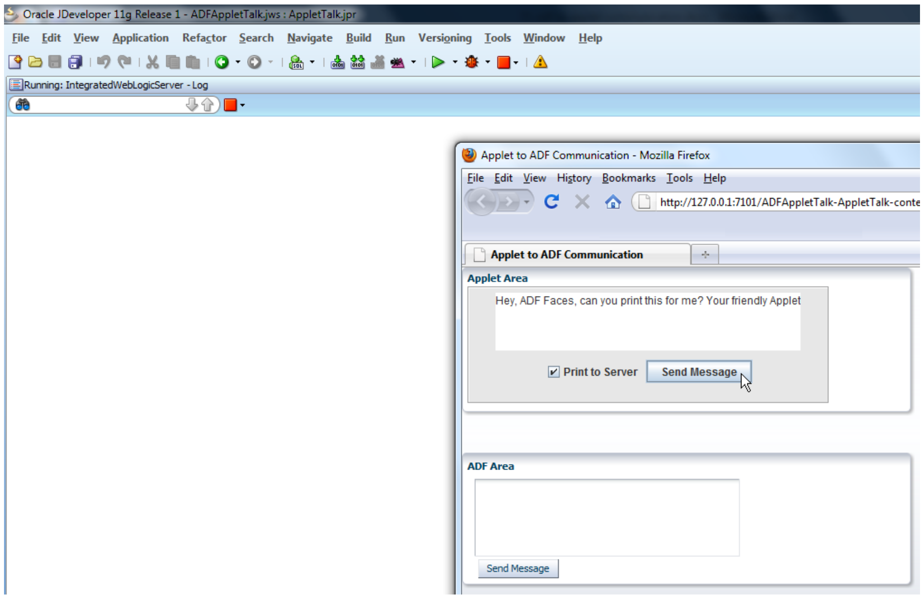
```
<af:inputText id="it1" rows="5"   columns="50" >
  <af:serverListener
          type="AppletIsCallingADF"  method="#{AppletCallCenter.onAppletCall}
   "/>
  <af:clientListener method="clearInputTextField"  type="dblClick"/>
</af:inputText>
```

Note: For JavaScript in ADF Faces to find the `af:inputText` component instance, the text field must have an `af:clientListener` attached or its *clientComponent* attribute set to **true**. If, like in the above case, the text field also has an `af:serverListener` attached, then it must also have an `af:clientListener` defined

ADF Faces managed bean to Applet communication

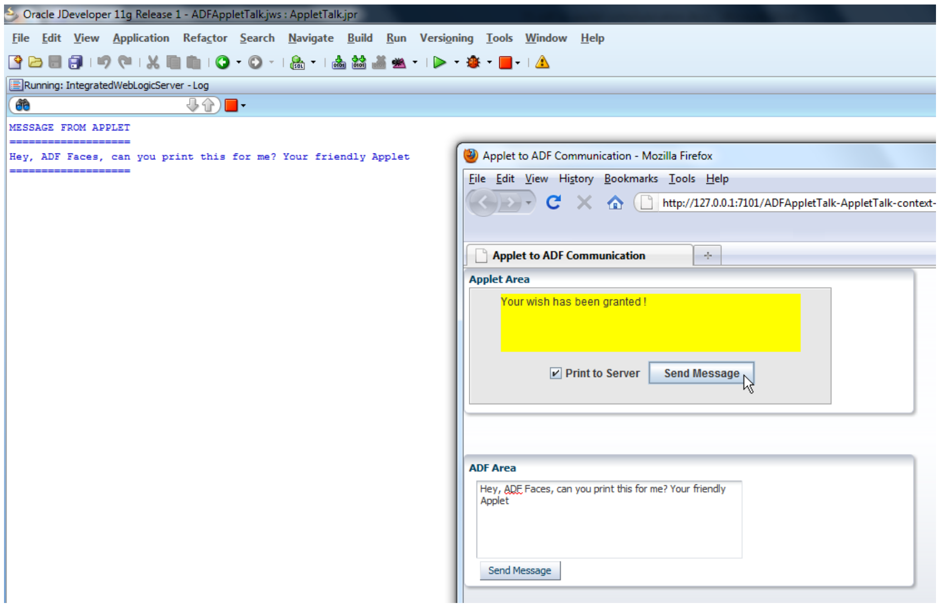
Using the Apache Trinidad `ExtendedRenderKitService` class, managed bean in ADF Faces can call client JavaScript, which then can call back to the JavaApplet. This way, messages can be sent from the server to the client side Applet.

Note: When communicating from a managed bean to the Java Applet, keep browser and network latency in mind. Especially when sending multiple messages, then, unless you handle queueing of server side messages, messages may get lost if the browser cannot handle the frequency you sent them.


Oracle JDeveloper 11g Release 1 - ADFAppletTalk.jws : AppletTalk.jpr
File Edit View Application Refactor Search Navigate Build Run Versioning Tools Window Help
Running: IntegratedWebLogicServer - Log
Applet to ADF Communication - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:7101/ADFAppletTalk-AppletTalk-conte
Applet to ADF Communication
Applet Area
Print to Server
Send Message
ADF Area
Send Message

Hey, ADF Faces, can you print this for me? Your friendly Applet

The image below shows a message sent from a managed bean in response to the Java Applet calling the server.


Oracle JDeveloper 11g Release 1 - ADFAppletTalk.jws : AppletTalk.jpr
File Edit View Application Refactor Search Navigate Build Run Versioning Tools Window Help
Running: IntegratedWebLogicServer - Log
Applet to ADF Communication - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:7101/ADFAppletTalk-AppletTalk-context-
Applet to ADF Communication
Applet Area
Print to Server
Send Message
ADF Area
Send Message

```
MESSAGE FROM APPLET
===================
Hey, ADF Faces, can you print this for me? Your friendly Applet
===================
```

Your wish has been granted !

Hey, ADF Faces, can you print this for me? Your friendly Applet

…

```
   //send feedback back to the Applet
    FacesContext fctx = FacesContext.getCurrentInstance();
    ExtendedRenderKitService erks =
           Service.getRenderKitService(fctx,
                                        ExtendedRenderKitService.class);
    erks.addScript(fctx,
     "window.writeAnyMessageToApplet('Your wish has been granted !')");
```

…

Configuring the Java Applet for deployment

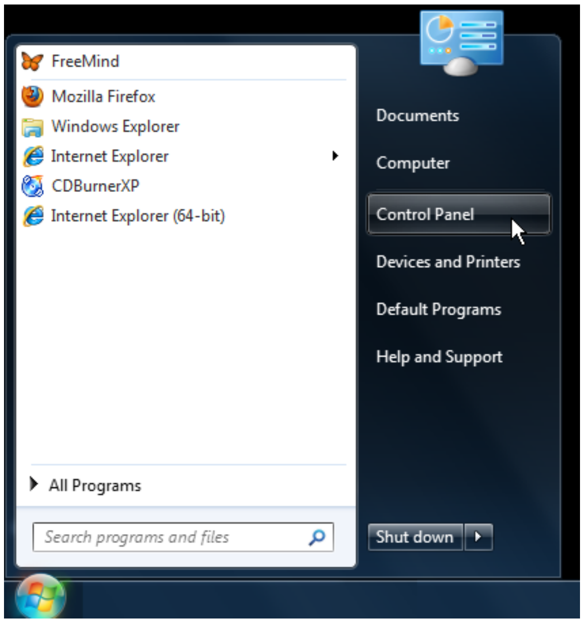
Java Applets are deployed with the application but downloaded to the client at runtime. This means that the Java Applet source, usually a Java Archive (JAR) file needs to be located in the Web project's public-html folder or a sub-folder of it. The JAR file is then referenced from the Applet *archive* tag

archive="/ADFAppletTalk-AppletTalk-context-root/messageApplet.jar"

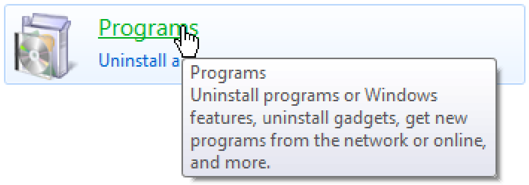
In the sample above, the archive reference to the Applet JAR file located in the *public-html* folder starts from the Java EE context root. The Java EE context root is determined by the Web project's **Java EE Application** property. You access this property by double clicking the Web project node in Oracle JDeveloper.

Ensuring the JavaPlugin is used

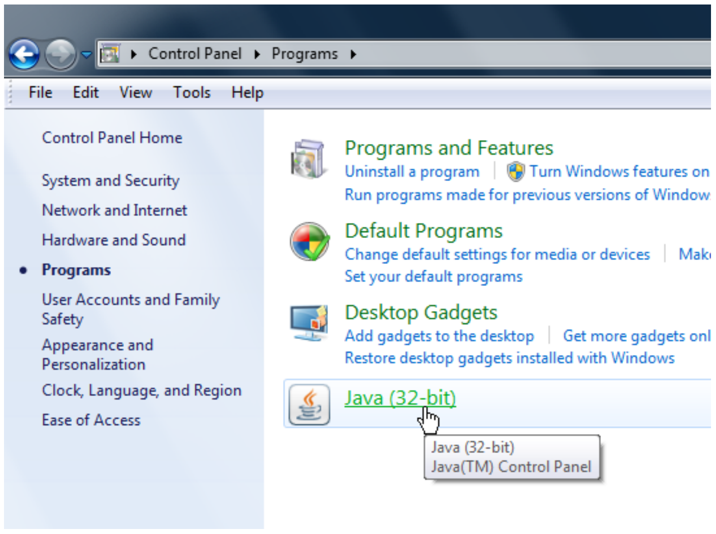
The one thing JavaScript and Java Applets have in common is that browser support varies. For Java Applets, the solution providing a consistent runtime environment is the Java Plugin that can be used instead of the browser native Java VM.



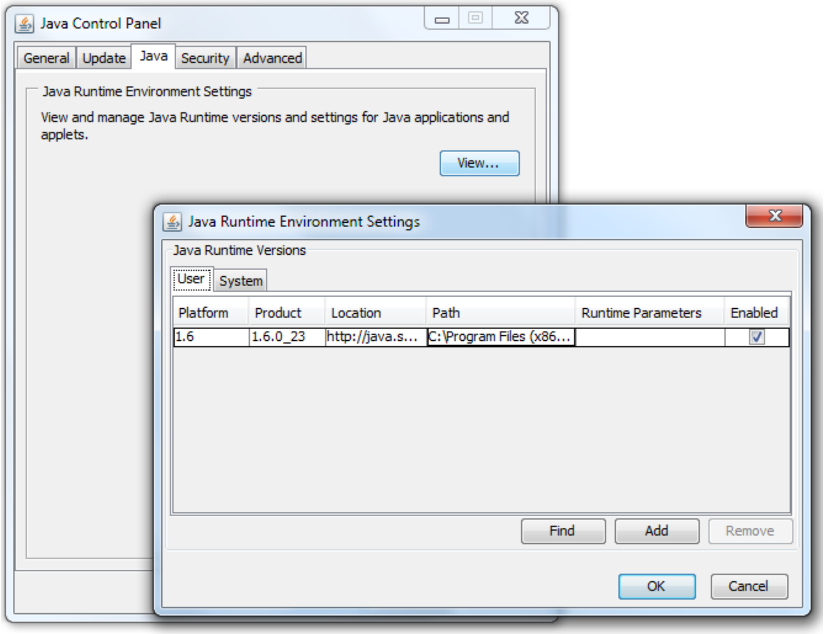
The Java Plugin is installed to replace the browser Virtual Machine when installing the Java Runtime Environment (JRE) to a computer. To test this on Windows, **select Start | Control Panel** and then click **Programs**


Programs
Uninstall a
Programs
Uninstall programs or Windows features, uninstall gadgets, get new programs from the network or online, and more.

Click the Java entry, as shown below, to access the configuration panel.



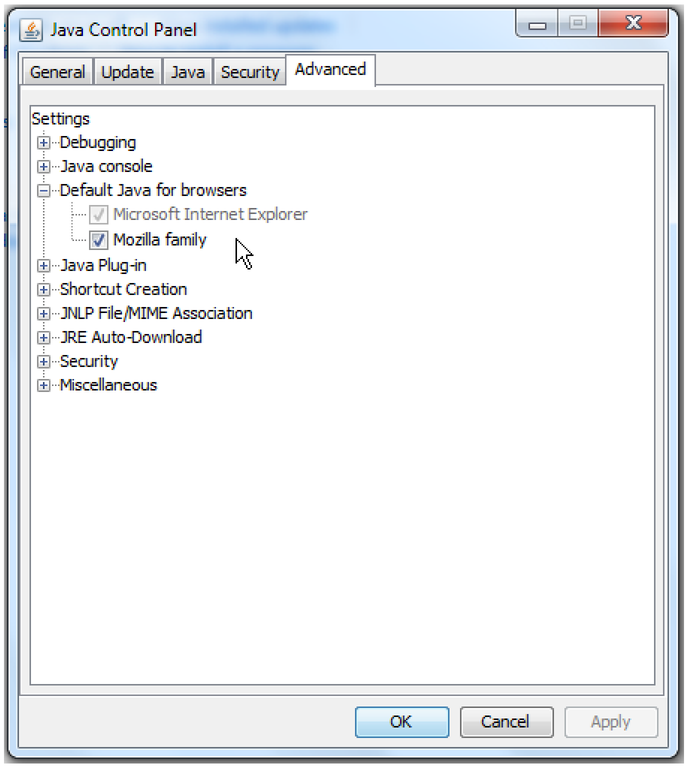
Select the **Java** tab to ensure the Java version that is installed. This information becomes handy in case you experience a Java minor or major version mismatch, which may happen when an Applet is compiled with an old version of Java.



View and manage Java Runtime versions and settings for Java applications and applets.

Platform	Product	Location	Path	Runtime Parameters	Enabled
1.6	1.6.0_23	http://java.s...	C:\Program Files (x86...		✓

Click the **Advanced** tab and expand the **Default Java for browsers** entry to see a list of browsers and their use of the Java Plugin at runtime.



Settings
- Debugging
- Java console
- Default Java for browsers
  - [✓] Microsoft Internet Explorer
  - [✓] Mozilla family
- Java Plug-in
- Shortcut Creation
- JNLP File/MIME Association
- JRE Auto-Download
- Security
- Miscellaneous

Note: Alternatively you can change the JSF page's Java Applet tag reference to use the **EMBED** and/or **OBJECTS** tag to enforce the use of the Java Plugin. In this case, the lookup of the Applet changes and the plugin also needs to have its *mayscript* property set to allow the Applet to send JavaScript to the client.

If your Java Applet requires access to the client computer, then, in addition to downloading the JAR file, the JAR file needs to be signed with a certificate that the user needs to accept. Signing JAR files is out of scope for this article, but there are plenty resources available on the Internet describing this process.

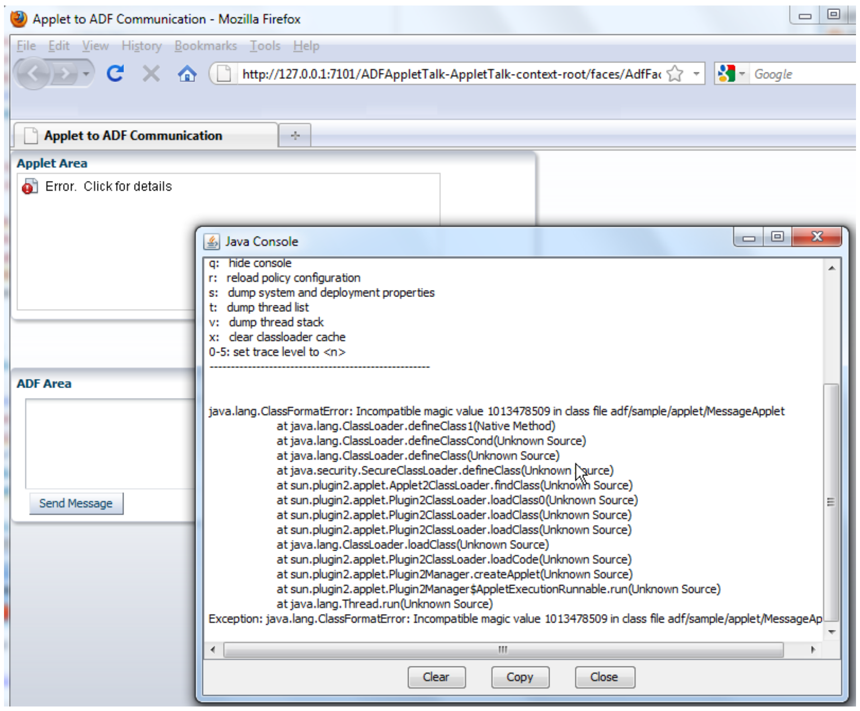
The infamous "incompatible magic value" issue

Not every error message makes sense, and some error messages cause more confusion than the help they are supposed to provide. The infamous "incompatible magic value" message has confused developers over many years.

Note: I don't want to imagine the cost caused by this misleading message as developers spent their time researching the real issue of the failed Java Applet loading instead of doing their job. The error message itself is shown below:

java.lang.ClassFormatError: **Incompatible magic value 1013478509** in class file adf/sample/applet/MessageApplet

at java.lang.ClassLoader.defineClass1(Native Method)
…


Applet to ADF Communication - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:7101/ADFAppletTalk-AppletTalk-context-root/faces/AdfFa
Google
Applet to ADF Communication
Applet Area
Error. Click for details
ADF Area
Send Message
Java Console
Clear
Copy
Close

```
q:  hide console
r:  reload policy configuration
s:  dump system and deployment properties
t:  dump thread list
v:  dump thread stack
x:  clear classloader cache
0-5: set trace level to <n>
----------------------------------------------------


java.lang.ClassFormatError: Incompatible magic value 1013478509 in class file adf/sample/applet/MessageApplet
	at java.lang.ClassLoader.defineClass1(Native Method)
	at java.lang.ClassLoader.defineClassCond(Unknown Source)
	at java.lang.ClassLoader.defineClass(Unknown Source)
	at java.security.SecureClassLoader.defineClass(Unknown Source)
	at sun.plugin2.applet.Applet2ClassLoader.findClass(Unknown Source)
	at sun.plugin2.applet.Plugin2ClassLoader.loadClass0(Unknown Source)
	at sun.plugin2.applet.Plugin2ClassLoader.loadClass(Unknown Source)
	at sun.plugin2.applet.Plugin2ClassLoader.loadClass(Unknown Source)
	at java.lang.ClassLoader.loadClass(Unknown Source)
	at sun.plugin2.applet.Plugin2ClassLoader.loadCode(Unknown Source)
	at sun.plugin2.applet.Plugin2Manager.createApplet(Unknown Source)
	at sun.plugin2.applet.Plugin2Manager$AppletExecutionRunnable.run(Unknown Source)
	at java.lang.Thread.run(Unknown Source)
Exception: java.lang.ClassFormatError: Incompatible magic value 1013478509 in class file adf/sample/applet/MessageAp
```

When you see this error message then, most likely the Java archive could not be found, in which case the server replies with http-404 or similar. The server side http-404 error message is delivered in an HTML document that the Java Plugin eagerly attempts to treat like a JAR file. Confused about the mismatch, the Plugin shows this confusing message. However, now you know that when the magic number shows incompatible, you need to double check the JAR or codebase setting (if not using JAR files for the Applet download).

Download

You can download the Java Applet and ADF Faces sources from the ADF Code Corner website. Its sample 71 and provided as Oracle JDeveloper workspaces

http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html

RELATED DOCOMENTATION

☒	Using JavaScript in ADF Faces applications – Whitepaper http://www.oracle.com/technetwork/developer-tools/jdev/1-2011-javascript-302460.pdf
☒	
☒	