

# Going Beyond Installation with JNLP\*

By John Zukowski

intel.®

**Introduction to JNLP ..... 3**  
**Getting Started ..... 3**  
**The JNLP API ..... 5**  
**The Source Code ..... 9**  
**Conclusion ..... 13**  
**Resources ..... 14**  
**About the Author ..... 14**

## Introduction to JNLP

The Java<sup>\*</sup> Network Launch Protocol (JNLP<sup>\*</sup>) is the standard Java deployment option for delivering rich client-side applications. Through implementations such as Sitraka's DeployDirector<sup>\*</sup> or the open source OpenJNLP<sup>\*</sup> effort, you can deploy Java applets and applications through the browser to be executed outside the browser. Once deployed, your program can be run without bringing up a browser or downloading the classes again, though the system will check for updates to the program.

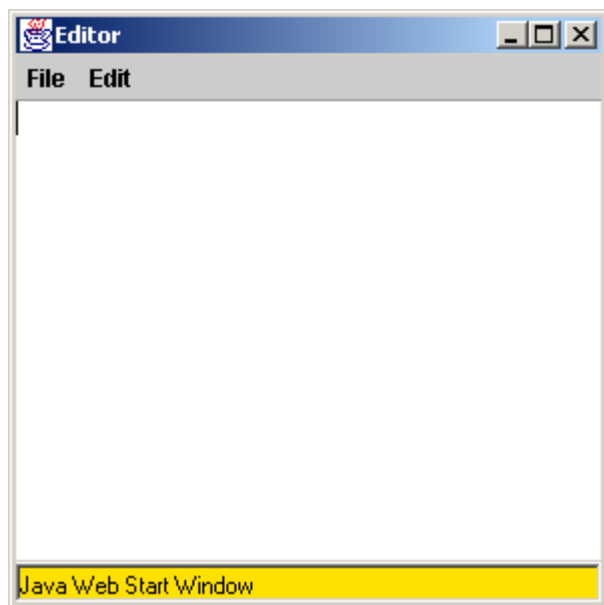
This paper discusses packaging Web applications using the `javax.jnlp` package, which defines the JNLP API. It provides a collection of classes and interfaces for working locally within the untrusted client environment provided by DeployDirector, OpenJNLP, Java Web Start<sup>\*</sup>, and others.

## Getting Started

The JNLP API is actually rather small. It consists of one class, twelve interfaces, and an exception. The class, **ServiceManager**, is used to look up services, which are a majority of the interfaces. Use these interfaces to access the system clipboard, read or write files, or print. Each attempted request triggers a security warning the user must confirm before the action succeeds. Standard Java code signing mechanisms also are supported for activities such as accessing a third-party Web server, but they are not part of the JNLP API.

## Creating a Text Editor

Before exploring the API, create a simple test application—a little text editor that permits saving locally on the client machine, reading files from the client machine, cut/copy/paste support from the system clipboard, and printing. The program consists of a JTextArea with a menu and will look like this:



Notice that since the program is unsigned, the app displays a yellow warning bar across the bottom, similar to running an unsigned applet.

Here is the source code for the user interface:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Editor extends JFrame {

    JTextArea textArea;

    public Editor() {
        super("Editor");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container contentPane = getContentPane();
        textArea = new JTextArea();
```

```

JScrollPane pane = new JScrollPane(textArea);
contentPane.add(pane, BorderLayout.CENTER);

JMenuBar bar = new JMenuBar();
setJMenuBar(bar);

JMenu file = new JMenu("File");
bar.add(file);

JMenuItem item = new JMenuItem("New");
ActionListener listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        textArea.setText("");
    }
};
item.addActionListener(listener);
file.add(item);

item = new JMenuItem("Open");
file.add(item);

item = new JMenuItem("Save");
file.add(item);

item = new JMenuItem("Print");
file.add(item);

item = new JMenuItem("Exit");
listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
};
item.addActionListener(listener);
file.add(item);

JMenu edit = new JMenu("Edit");
bar.add(edit);

item = new JMenuItem("Cut");
edit.add(item);

item = new JMenuItem("Copy");
edit.add(item);

item = new JMenuItem("Paste");
edit.add(item);

setSize(300, 300);
}

public static void main(String args[]) {
    new Editor().show();
}
}

```

Feel free to add menu mnemonics or other convenience features as you wish to this functional interface.

## Packaging

The next step is to package the application for Java Web Start (or whichever JNLP implementation you use). Since Java Web Start is installed automatically on Windows\* with the Java 2 SDK, version 1.4 release (and separately installable for Linux\* and Solaris\* users), we'll use it for this exercise.

Packaging for Java Web Start involves creating a JAR file with the program's classes and creating an XML file (named with a `jnlp` extension) that describes the application. Opening the JNLP file through the browser kicks off Java Web Start.

Assuming you are working in a directory just for this project, create the JAR file with the following command:

```
jar cvf Editor.jar *.class
And here are the contents of Editor.jnlp.
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+"
  codebase="file:///c:/jzventures/intel/jnlp/"
>
<information>
  <title>Text Editor</title>
  <vendor>Java Developer Center</vendor>
  <homepage href="http://java.sun.com/jdc" />
  <description>JNLP Editor</description>
</information>
<offline-allowed/>
<resources>
  <j2se version="1.4+" />
  <jar href="Editor.jar"/>
</resources>
<application-desc main-class="Editor" />
</jnlp>
```

Be sure that the **codebase** matches the location from which you are loading the file. If you are placing the application on a Web server, replace the **file:** URL with an **http:** URL. Another condition of placing the application on a Web server is that the server must report the proper MIME type for the application loader. For Tomcat\* 4.0 and many of the latest releases of Web and application servers, this is preconfigured. Otherwise, be sure to map the `.jnlp` extension to the `application/x-java-jnlp-file` MIME type. How to do this is dependent on the server you are using.

Load the `Editor.jnlp` file in your browser to bring up the application interface shown previously.

## The JNLP API

With the test program framework created and packaged, it's time to examine the actual JNLP API. As previously mentioned, the API is found in the `javax.jnlp` package. The classes for this package are found in the `javaws.jar` file that comes with Java Web Start. For Windows users, the default installation directory is `C:\Program Files\Java Web Start`. After adding the `javaws.jar` file to your CLASSPATH, you can create programs that use the JNLP API. It is important to point out that once you use the JNLP API, your program is locked into running within one of the JNLP implementations like Java Web Start. The services offered through JNLP require the appropriate runtime container in order to execute.

The JNLP API is comprised of eight services, each with its own interface:

- BasicService
- ClipboardService
- DownloadService
- ExtensionInstallerService
- FileOpenService
- FileSaveService
- PersistenceService
- PrintService
- 

Implementations of these interfaces offer access to the client machine. When such access is needed, even in an untrusted execution environment, the user is prompted for permission before access is granted. The prompt isn't for global access but granted on a very restricted basis. For instance, with `FileOpenService`, the user is granted access to read a file on the client machine but is restricted to selecting the file from a file dialog window.

The `ServiceManager` class controls use of the JNLP services. Use the `lookup()` method of the `ServiceManager` to lookup a service by fully qualified interface name, and an implementation is returned for you to use:

```
FileOpenService fileOpenService = (FileOpenService)
  ServiceManager.lookup("javax.jnlp.FileOpenService");
```

In the case of the `FileOpenService`, a single or multi-select file dialog box is shown. After selection, you can get the `InputStream` associated with the selection to read. The following security prompt appears prior to the dialog:



Let's add this behavior to the text editor to demonstrate.

After adding javaws.jar to your CLASSPATH, import the necessary package:

```
import javax.jnlp.*;
```

Then create and add the appropriate listener. Look up the appropriate service within the listener. Next, use the service returned to prompt for an open file selection. In this case, the method is `openFileDialog()`, which automatically deals with the security advisory. This then returns a `FileContents` object from which to read the contents to display in the text area.

Here's what the whole listener method looks like:

```
listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            // Lookup service
            FileOpenService fileOpenService =
                (FileOpenService)ServiceManager.lookup(
                    "javax.jnlp.FileOpenService");
            // Request file to open
            String path = ".";
            String extensions[] = {"txt", "java", "log"};
            FileContents contents =
                fileOpenService.openFileDialog(
                    path, extensions);
            // Get stream to selected file
            InputStream is = contents.getInputStream();
            // Read in file
            InputStreamReader isr =
                new InputStreamReader(is);
            BufferedReader reader = new BufferedReader(isr);
            StringWriter sw = new StringWriter();
            PrintWriter writer = new PrintWriter(sw);
            String line;
            while ((line = reader.readLine()) != null) {
                writer.println(line);
            }
            // Show contents in text area
            textArea.setText(sw.toString());
            // Close streams
            writer.close();
            reader.close();
        } catch (UnavailableServiceException use) {
            System.err.println("Service Unavailable: "
                + use);
        } catch (IOException ioe) {
            System.err.println("I/O Problems: " + ioe);
        }
    }
}
```

```

    }
}
};
item.addActionListener(listener);

```

Once you've added this to the `Editor.java` file, recompile the file and remember to recreate the JAR file before running the application. Depending upon your platform, you may notice that the file dialog can opt to not offer the use of the file extensions provided.

Working our way through the remaining "File" menu options, saving a file is similar to reading, but is actually simpler. You don't actually have to write the data, only provide an `InputStream` that contains the bytes to save. Here's the core part of that code:

```

// Get contents to save as bytes
String text = textArea.getText();
byte bytes[] = text.getBytes();

// Convert to InputStream
ByteArrayInputStream input =
    new ByteArrayInputStream(bytes);

// Lookup service
FileSaveService fileSaveService = (FileSaveService)
    ServiceManager.lookup("javax.jnlp.FileSaveService");

// Request file to save
String path = ".";
String extensions[] = {"txt", "java", "log"};
String dummyFilename = "dummy.txt";
FileContents contents =
    fileSaveService.saveFileDialog(
        path, extensions, input, dummyFilename);
if (contents != null) {
    System.out.println("Saved to: " +
        contents.getName());
}
}

```

All of the menu options work in a similar fashion: look up the service, and then use it. In the case of the `PrintService`, you must implement the `java.awt.print.Printable` or `java.awt.print.Pageable` interfaces. As this isn't an exercise in printing with Java, you'll provide a simple implementation that just prints a screen dump of the `JTextArea`. If you want to paginate the text, you'll find that the `java.awt.font.TextLayout` and `java.awt.font.LineBreakMeasurer` classes are valuable.

```

import java.awt.print.*;
. . .
public class Editor extends JFrame
    implements Printable {
. . .
    public int print(Graphics g,
        PageFormat pageFormat, int pageIndex)
        throws PrinterException {
        int x = (int)pageFormat.getImageableX();
        int y = (int)pageFormat.getImageableY();
        g.translate(x, y);
        if (pageIndex == 0) {
            textArea.paint(g);
            return Printable.PAGE_EXISTS;
        } else {
            return Printable.NO_SUCH_PAGE;
        }
    }
}
}

```

The actual printing code behind the menu is just look up `PrintService` and call `print()`, passing in the `Printable` implementor.

```
// Lookup service
PrintService printService = (PrintService)
    ServiceManager.lookup("javax.jnlp.PrintService");

// Print screen image
boolean status = printService.print(Editor.this);
if (!status) {
    System.err.println("Problems printing");
}
```

That leaves the Edit menu and the Cut, Copy, and Paste operations. Here, the service of interest is the `ClipboardService`. Other than looking up the service, the rest is just an understanding of clipboard access and the `java.awt.datatransfer` package.

The Cut and Copy operations are the same as far as the `ClipboardService` goes:

```
// Put on Clipboard
String selection = textArea.getSelectedText();
StringSelection data = new StringSelection(selection);
clipboardService.setContents(data);
```

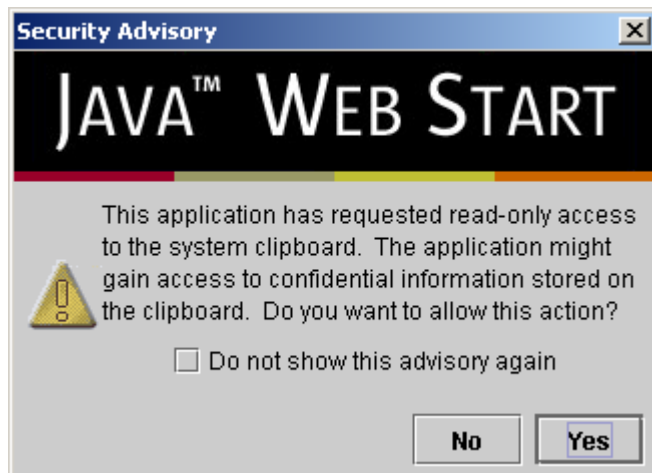
The Paste operation requires a little bit of work to get the right `DataFlavor`.

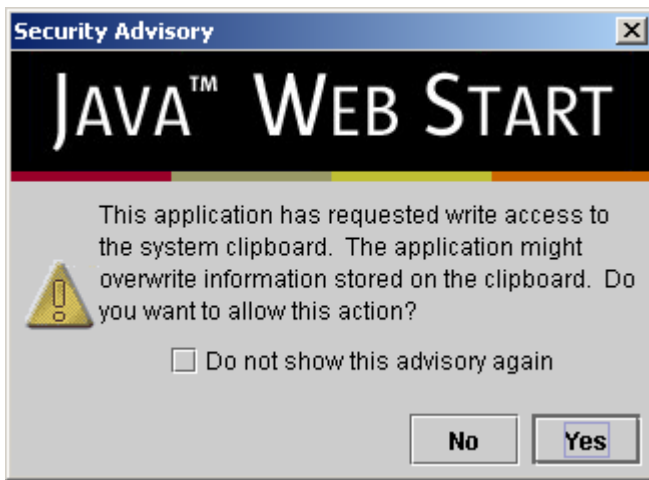
```
// Get from Clipboard
Transferable clipData =
    clipboardService.getContents();
if (clipData != null) {
    if (clipData.isDataFlavorSupported(
        DataFlavor.stringFlavor)) {
        String string = (String)(clipData.getTransferData(
            DataFlavor.stringFlavor));
        // Put in JTextArea
        textArea.replaceSelection(string);
    }
}
```

In both cases, just be sure to import the appropriate package:

```
import java.awt.datatransfer.*;
```

Depending upon the operation performed, the prompt to the user indicates whether read-only or write access is desired:





Unless the user checks “Do not show this advisory again,” every access to the system clipboard will require confirmation from the security advisory.

## The Source Code

Here is the complete source for the program.

### Editor.java

```
import java.awt.*;
import java.awt.datatransfer.*;
import java.awt.event.*;
import java.awt.print.*;
import java.io.*;
import javax.jnlp.*;
import javax.swing.*;

public class Editor extends JFrame
    implements Printable {

    JTextArea textArea;

    public Editor() {
        super("Editor");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container contentPane = getContentPane();
        textArea = new JTextArea();
        JScrollPane pane = new JScrollPane(textArea);
        contentPane.add(pane, BorderLayout.CENTER);

        JMenuBar bar = new JMenuBar();
        setJMenuBar(bar);

        JMenu file = new JMenu("File");
        bar.add(file);

        JMenuItem item = new JMenuItem("New");
        ActionListener listener = new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                textArea.setText("");
            }
        };
        item.addActionListener(listener);
        file.add(item);

        item = new JMenuItem("Open");
        listener = new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    // Lookup service
```

```

FileOpenService fileOpenService =
    (FileOpenService)ServiceManager.lookup(
        "javax.jnlp.FileOpenService");

// Request file to open
String path = ".";
String extensions[] =
    {"txt", "java", "log"};
FileContents contents =
    fileOpenService.openFileDialog(
        path, extensions);

// Get stream to selected file
InputStream is = contents.getInputStream();

// Read in file
InputStreamReader isr =
    new InputStreamReader(is);
BufferedReader reader =
    new BufferedReader(isr);
StringWriter sw = new StringWriter();
PrintWriter writer = new PrintWriter(sw);
String line;
while ((line = reader.readLine()) != null) {
    writer.println(line);
}

// Show contents in text area
textArea.setText(sw.toString());

// Close streams
writer.close();
reader.close();

} catch (UnavailableServiceException use) {
    System.err.println("Service Unavailable: "
        + use);
} catch (IOException ioe) {
    System.err.println("I/O Problems: " + ioe);
}
}
};
item.addActionListener(listener);
file.add(item);

item = new JMenuItem("Save");
listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            // Get contents to save as bytes
            String text = textArea.getText();
            byte bytes[] = text.getBytes();

            // Convert to InputStream
            ByteArrayInputStream input =
                new ByteArrayInputStream(bytes);

            // Lookup service
            FileSaveService fileSaveService =
                (FileSaveService)ServiceManager.lookup(
                    "javax.jnlp.FileSaveService");

            // Request file to save
            String path = ".";
            String extensions[] =
                {"txt", "java", "log"};

```

```

String dummyFilename = "dummy.txt";
FileContents contents =
    fileSaveService.saveFileDialog(
        path, extensions, input, dummyFilename);
if (contents != null) {
    System.out.println("Saved to: "
        + contents.getName());
}

} catch (UnavailableServiceException use) {
    System.err.println("Service Unavailable: "
        + use);
} catch (IOException ioe) {
    System.err.println("I/O Problems: " + ioe);
}
}
};
item.addActionListener(listener);
file.add(item);

item = new JMenuItem("Print");
listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            // Lookup service
            PrintService printService =
                (PrintService)ServiceManager.lookup(
                    "javax.jnlp.PrintService");

            // Print screen image
            boolean status =
                printService.print(Editor.this);
            if (!status) {
                System.err.println("Problems printing");
            }

        } catch (UnavailableServiceException use) {
            System.err.println("Service Unavailable: "
                + use);
        }
    }
};
item.addActionListener(listener);
file.add(item);

item = new JMenuItem("Exit");
listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
};
item.addActionListener(listener);
file.add(item);

JMenu edit = new JMenu("Edit");
bar.add(edit);

item = new JMenuItem("Cut");
listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            // Lookup service
            ClipboardService clipboardService =
                (ClipboardService)ServiceManager.lookup(
                    "javax.jnlp.ClipboardService");

```

```

// Put on Clipboard
String selection =
    textArea.getSelectedText();
StringSelection data =
    new StringSelection(selection);
clipboardService.setContents(data);

// Remove selection from JTextArea
textArea.replaceSelection("");

} catch (UnavailableServiceException use) {
    System.err.println("Service Unavailable: "
        + use);
}
}
};
item.addActionListener(listener);
edit.add(item);

item = new JMenuItem("Copy");
listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            // Lookup service
            ClipboardService clipboardService =
                (ClipboardService)ServiceManager.lookup(
                    "javax.jnlp.ClipboardService");

            // Put on Clipboard
            String selection =
                textArea.getSelectedText();
            StringSelection data =
                new StringSelection(selection);
            clipboardService.setContents(data);

        } catch (UnavailableServiceException use) {
            System.err.println("Service Unavailable: "
                + use);
        }
    }
};
item.addActionListener(listener);
edit.add(item);

item = new JMenuItem("Paste");
listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            // Lookup service
            ClipboardService clipboardService =
                (ClipboardService)ServiceManager.lookup(
                    "javax.jnlp.ClipboardService");

            // Get from Clipboard
            Transferable clipData =
                clipboardService.getContents();
            if (clipData != null) {
                if (clipData.isDataFlavorSupported(
                    DataFlavor.stringFlavor)) {
                    String string = (String)
                        (clipData.getTransferData(
                            DataFlavor.stringFlavor));
                    // Put in JTextArea
                    textArea.replaceSelection(string);
                }
            }
        }
    }
};

```

```

    } catch (UnavailableServiceException use) {
        System.err.println("Service Unavailable: "
            + use);
    } catch (UnsupportedFlavorException ufe) {
        System.err.println("Bad flavor: " + ufe);
    } catch (IOException ioe) {
        System.err.println("I/O problems: " + ioe);
    }
}
};
item.addActionListener(listener);
edit.add(item);

setSize(300, 300);
}

public int print(Graphics g,
    PageFormat pageFormat, int pageIndex)
    throws PrinterException {
    int x = (int)pageFormat.getImageableX();
    int y = (int)pageFormat.getImageableY();
    g.translate(x, y);
    if (pageIndex == 0) {
        textArea.paint(g);
        return Printable.PAGE_EXISTS;
    } else {
        return Printable.NO_SUCH_PAGE;
    }
}

public static void main(String args[]) {
    new Editor().show();
}
}

```

### Editor.jnlp

```

<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+"
    codebase="file:///c:/jzventures/intel/jnlp/"
>
<information>
    <title>Text Editor</title>
    <vendor>Java Developer Center</vendor>
    <homepage href="http://java.sun.com/jdc" />
    <description>JNLP Editor</description>
</information>
<offline-allowed/>
<resources>
    <j2se version="1.4+" />
    <jar href="Editor.jar"/>
</resources>
<application-desc main-class="Editor" />
</jnlp>

```

Note: If you want debug messages printed to `System.out/err`, enable the Java Console from the application manager specific to your JNLP implementation. For Java Web Start, this is available from <http://java.sun.com/products/javawebstart/demos.html>.

## Conclusion

The JNLP API is a small but powerful API to add to your bag of programming tricks. Through the simple service lookup method, you're given access from an untrusted program directly to the end user's desktop, without the security risk. While code signing is certainly available through Java Web Start, for many tasks, the associated hassles are not required given the controlled limited access available through the classes of the `javax.jnlp` package.

## Resources

[JNLP API javadoc](#)\*\*

[Java Web Start](#)\*\*

[Java Web Start Developer's Guide](#)\*\*

[DeployDirector](#)\*\*

[OpenJNLP](#)\*\*

[Tomcat](#)\*\*

## About the Author



John Zukowski serves as the resident guru for a number of jGuru's community-driven FAQs (<http://www.jguru.com/>\*\*) and is a popular Java columnist around the Internet. His latest books are *Learn Java with JBuilder 6* (Apress, 2002)

(<http://www.amazon.com/exec/obidos/ASIN/1893115984/ref=nosim/johnzukowshomefo/>\*\*)

and *Mastering Java 2: J2SE v. 1.4* (Sybex, 2002) (<http://www.amazon.com/exec/obidos/ASIN/078214022X/ref=nosim/johnzukowshomefo/>\*\*).

\*\* Intel provides Internet links in this document as a convenience to its customers. The linked sites are independent of Intel and Intel does not warrant and cannot be responsible for their contents.