Resource Loading Tutorial and Rachel Open-Source Toolkit for Java Web Start/JNLP

Gerald Bauer

September, 29th, 2001

About This Booklet

Web Start demands that you load all resources (such as property files, images, html pages) from jars because Web Start can only ship jars to the client desktop. Moreover, you can't use absolute or relative file paths to locate your jars holding your resources (e.g. `jar:file:///c:/java/jws/.cache/images/save.gif`). Absolute file paths won't work because you never know where Web Start will put your jars on the user's machine. Relative file paths won't work because Web Start mangles the names of your jars (e.g. `venus.jar` becomes `RMvenus.jar`) and every JNLP client implementation has the right to mangle your names in a different way and you, therefore, can't predict the name with which your jar will be rechristend and end up on the user's machine in the application cache.

The only option left is using class loaders to get your resources out of jars. The Resource Loading Tutorial [`http://www.geocities.com/vamp201/tutorial.html`] explains how to load images and properties from a jar by building a real-world example that you can kick off at your very own desktop. Moreover, it reveals the resource anchor trick that always puts the right classloader in your hands.

The Rachel Resource Loading Toolkit User Documentation [`http://www.geocities.com/vamp201/resources.html`] gets you on a higher plane and shows you how easy resource loading can be by wrapping the low-level classloader trickery in a custom URL protocol handler. Rachel's `class://` URL protocol handler allows you to load images, property files, theme packs or any other resources using Java's standard `java.net.URL` class. You can even load resources dynamically by loading resources listed in property files that were themself loaded from a jar or you can use Java's built-in browser to display HTML pages retrieved from jars. Rachel's multi-threaded, ultra leight-weight web server also allows you to display HTML pages retrieved from jars in external household browsers such as Mozilla, Netscape, or Opera.

The latest version of the Resource Loading Tutorial resides at Venus Application Publisher's (Vamp) website at `http://www.geocities.com/vamp201/tutorial.html`. The latest version of the Rachel Open Source Resource Loading Toolkit User Documentation resides at `http://www.geocities.com/vamp201/resources.html`. For both documents you also find the original DocBook, HTML and other formats there.

Your comments, suggestions and corrections are welcome. Send them to vamp201@yahoo.com. Note, that I cannot answer your individual resource loading questions. Instead post your questions at Sun's Java Web Start/JNLP forum [`http://forum.java.sun.com/forum.jsp?forum=38`] and I will answer them if I have a clue and time permits.

Resource Loading Tutorial

Overview

In this tutorial I create a real-world example that loads resources (e.g. properties and images) from a jar. You can kick off the Hello Venus example app at your very own desktop using Java Web Start.

- Browse the Source [`http://www.geocities.com/vamp201/tutorial/crossref/index.html`]
- Launch Hello Venus Resource Loading Tutorial Example [`http://www.jenomics.de/vamp/tutorial.jnlp`]
- Download Resource Loading Tutorial [`http://www.geocities.com/vamp201/download/tutorial.zip`]

First Impression

Here is a first impression of the Hello Venus resource loading example. The icons are retrieved from the jar as well as `menu.properties` which is used to create the help menu.

You can find all Hello Venus source files including `menu.properties` in the appendix.

Resource Anchor Trick Revealed

Retrieving resources from jars isn't hard if you use the resource anchor trick presented here. Start with creating an empty class and package it with your resources. Example:

```
public class ResourceAnchor
{
    public ResourceAnchor()
    {
    }
}
```

The `ResourceAnchor` class is just here to identify the Java archive that holds your resources. There is no need to burn the weather report on your toast in the constructor.

Here is the content of `tutorial.jar` that holds `ResourceAnchor.class` and all resources we want to retrieve such as `menu.properties`, `inform.gif` and `world2.gif`:

```
vamp/tutorial/CmdShowUrl.class
vamp/tutorial/HelloVenus$1.class
vamp/tutorial/HelloVenus.class
vamp/tutorial/Main.class
vamp/tutorial/menu.properties
vamp/tutorial/ResourceAnchor.class
vamp/tutorial/WebBrowser.class
vamp/tutorial/images/inform.gif
vamp/tutorial/images/world2.gif
```

To retrieve your resources you need to get a hand on a class loader. Here's the pick-up line:

```
ClassLoader cl = ResourceAnchor.class.getClassLoader();
```

Now, you are ready to drill for bits. Here is how you get an icon from `tutorial.jar`:

```
URL worldIconUrl = cl.getResource(
                     "vamp/tutorial/images/world2.gif" );
ImageIcon worldIcon = new ImageIcon( worldIconUrl );
```

Or in compressed form:

```
ImageIcon worldIcon =
  new ImageIcon( cl.getResource( "vamp/tutorial/images/world2.gif" ) );
```

Tip: Use only lower case resource names as on some machines retrieving resources with both upper and lower case letters will fail (e.g rename `World2.gif` to `world2.gif`).

And here is how to get `menu.properties` from `tutorial.jar`:

```
Properties props = new Properties();
try
{
  props.load(
    cl.getResourceAsStream( "vamp/tutorial/menu.properties" ));
}
catch( IOException ioex )
{
  System.err.println( "*** failed to load properties: " + ioex.toString() );
}
```

That's it. That's all there is to it.

Rachel - Open-Source Resource Loading Toolkit for Web

Start/JNLP

What is Rachel?

Rachel is Vamp's open-source resource loading toolkit for Java Web Start that makes resource loading easy again. There is no need to use low-level, sub-atomic, complex code as illustrated by the following example taken from Sun's Java Web Start Developer's Guide [`http://java.sun.com/products/javawebstart/docs/developersguide.html`]:

```
// Get current classloader
ClassLoader cl = this.getClass().getClassLoader();
// Create icons
Icon saveIcon = new ImageIcon( cl.getResource( "images/save.gif" ));
Icon cutIcon  = new ImageIcon( cl.getResource( "images/cut.gif" ));
```

Why all this messing around with `ClassLoader`'s and `getResource`? Isn't there a simpler way? Why is resource loading so hard for JavaWebStart software in the first place? What's the deal? How are Java Web Start application different from plain-vanilla Java desktop applications?

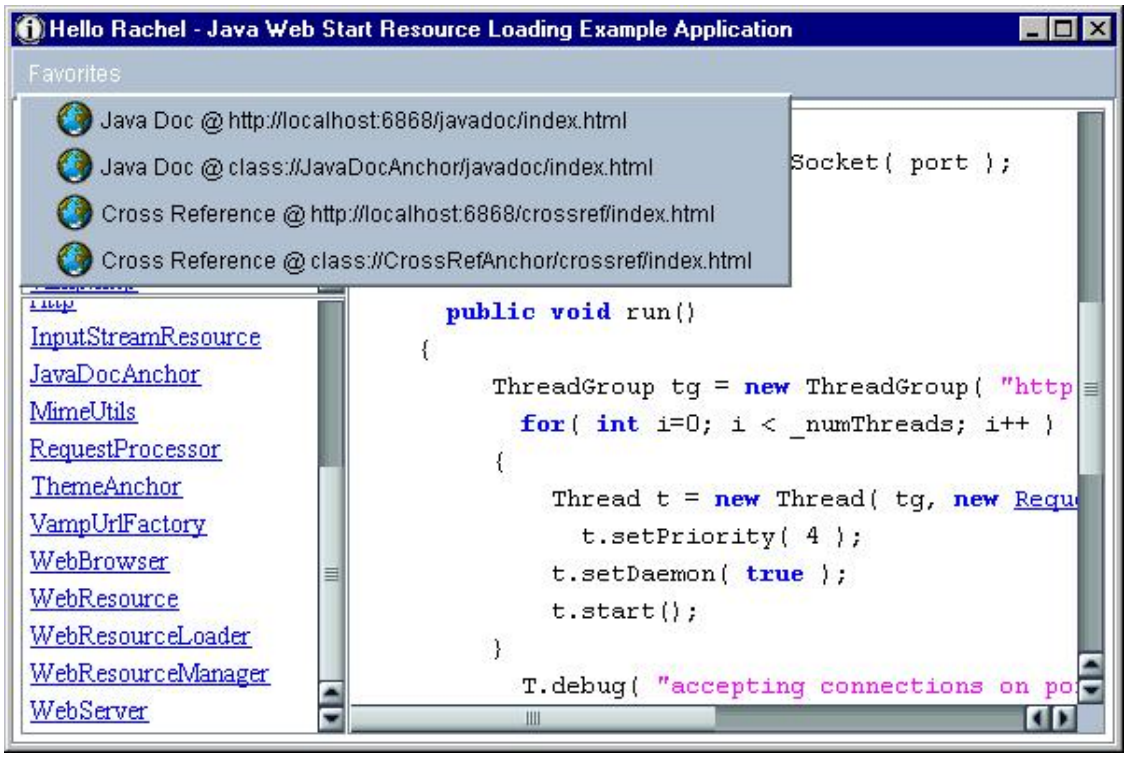
Here is the essence extracted from the Application Development Considerations section in Sun's Java Web Start Developer's Guide [`http://java.sun.com/products/javawebstart/docs/developersguide.html`] to justify this code:

Rule 1: Java Archives only. No loose files. All your resources have to be packaged in Java Archives (jar) if you want to have them delivered to the user's machine and kept up-to-date automatically by Java Web Start.

Rule 2: No file paths. You can't use absolute or relative file paths to locate your resources in jars (e.g. `jar:file:///c:/java/jws/.cache/images/save.gif`). Absolute file paths won't work because you never know where Java Web Start will put your jar on the user's machine. Relative file paths won't work because Java Web Start mangles the names of your jars (e.g. `venus.jar` becomes `RMvenus.jar`) and every JNLP client implementation has the right to mangle your names in a different way and you, therefore, can't predict the name with which your jar will be rechristend and end up on the user's machine in the application cache.

First Impression - Rachel in Action

Rachel's example app displays HTML pages packed up in `crossref.jar` and `javadoc.jar` in internal and external browsers and uses the Skin Look & Feel [`http://www.l2fprod.com`] 's Modern theme packed up in `themes.jar` thanks to Rachel's light-weight web server and `class://` URL protocol handler.



Resource Loading Made Easy

Rachel offers two solutions that make resource loading easy again. You won't miss the old plain-vanilla Java days as Java Web Start now also automatically up-dates your resources along with your class files.

Solution 1 installs a URL handler for a new protocol called `class://` that delivers content from jars identified by a Java class. Example:

```
class://ThemeAnchor/skinlf/themes/modern.zip
```

The Java class `ThemeAnchor` identifies the Java Archive `themes.jar` in the example above. There is no need to use absolute or relative file paths to identify jars. The fully qualified name of a single Java class in the jar is sufficient to find it in the application cache. To clear things up, here is the content of `themes.jar` which is part of the example application shipped with Rachel:

```
skinlf/themes/modern.zip
images/world2.gif
images/inform.gif
META-INF/Manifest.mf
ThemeAnchor.class
```

Once the URL handler is installed, resources can be loaded using the standard `java.net.URL` class. Example:

```
URL appIconUrl = new URL( "class://ThemeAnchor/images/inform.gif );
ImageIcon appIcon = new ImageIcon( appIconUrl );
```

Solution 2 embeds a multi-threaded ultra light-weight web server in your app that serves up content from jars in the Java Web Start cache. Your end-users can also access your resources (such as HTML documents) through external browsers using standard web addresses (e.g. `http://localhost:5454/crossref/index.html`) as long as your application is up and running (or the document lingers on in the browser's cache).

Method 1: Use class:// URL handler

To use a URL using the new protocol `class://` that delivers content from jars follow these steps:

Step 1: Register a new URL handler for the class:// protocol

Example:

```
URL.setURLStreamHandlerFactory( new VampUrlFactory() );
```

Step 2: Create a URL using the class:// protocol for the resource you want to retrieve from a jar

Use the following syntax when you create a URL for the `class://` protocol:

```
class://<class>/<path>
```

Examples:

```
class://venus.Tool/images/powered-by-velocity.gif
class://Chrome/skinlf/themes/aqua.zip
class://JavaDocAnchor/javadoc/index.html
class://ThemeAnchor/images/world2.gif
```

Step 3: Get resource using java.net.URL

To get a resource (e.g. an image, HTML document or theme pack) you can use Java's standard `java.net.URL` class. Example:

```
URL appIconUrl = new URL( "class://ThemeAnchor/images/inform.gif" );
ImageIcon appIcon =  new ImageIcon( appIconUrl );
```

or

```
URL themepack = new URL( "class://ThemeAnchor/skinlf/themes/modern.zip" );
Skin skin = SkinLookAndFeel.loadThemePack( themepack );
```

Tip: You can use this method to serve up HTML pages from any jar in your internal web browser (aka `JEditorPane`). Example:

```
URL crossRefUrl = new URL( "class://CrossRefAnchor/crossref/index.html" );
```

```
JEditorPane browser = new JEditorPane();
browser.setEditable( false );
browser.setContentType( "text/html" );
browser.setPage( crossRefUrl );
```

Method 2: Embed a multi-threaded, ultra light-weight web server into your app

To use Rachel's embedded multi-threaded ultra light-weight web server that serves up content from jars follow these steps:

Step 1: Package all your resources (documents, graphics, etc.) in a jar.

Step 2: Add an anchor class to every jar holding resources

Add an resource anchor class to every jar from which you want the embedded web server to serve up your resources. Example:

```
public class CrossRefAnchor
{
  public CrossRefAnchor() {}
}
```

Yes, that's it. The class is just here to identify the Java Archive. There is no need to calculate a random seed for your digital certificate or to burn the weather report on your toast.

Step 3: Add ClassResourceLoader to WebResourceManager for every jar holding resources

Example:

```
WebResourceManager roots = WebResourceManager.getInstance();
roots.addResourceLoader( new ClassResourceLoader( CrossRefAnchor.class ) );
roots.addResourceLoader( new ClassResourceLoader( JavaDocAnchor.class ) );
```

Step 4: Startup the multi-threaded ultra light-weight web server

Example:

```
try
{
  WebServer http = new WebServer( 7272, roots );
  http.start();
}
catch( IOException ioex )
{
  T.error( "*** failed to start http service: " + ioex.toString() );
}
```

Step 5: You are ready to roll

Display page in external or internal browser as you desire as Rachel is at your service. Example:

```
URL crossRefUrl = new URL( "http://localhost:7272/crossref/index.html" );
JEditorPane browser = new JEditorPane();
browser.setEditable( false );
browser.setContentType( "text/html" );
browser.setPage( crossRefUrl );
```

Rachel Links

- Browse Rachel's Source Code [http://www.geocities.com/vamp201/rachel/crossref/index.html]
- Browse Rachel's Java Doc [http://www.geocities.com/vamp201/rachel/javadoc/index.html]
- Download Rachel [http://www.geocities.com/vamp201/download/rachel.zip]
- Launch Rachel Example App [http://www.jenomics.de/vamp/rachel.jnlp]

Appendix

Hello Venus Resource Loading Tutorial Example Code

Main.java

```
package vamp.tutorial;

public class Main
{
   public static void main( String argv[] )
   {
       HelloVenus gui = new HelloVenus();
   }
}
```

menu.properties

```
help.menu: tutorial faq home
tutorial.text: Java Web Start Resource Loading Tutorial
tutorial.url:  http://www.geocities.com/vamp201/tutorial.html
faq.text: Unofficial Java Web Start/JNLP FAQ
faq.url: http://www.geocities.com/vamp201/jwsfaq.html
home.text: Venus Application Publisher (Vamp) Home
home.url: http://www.geocities.com/vamp201
```

ResourceAnchor.java

```
package vamp.tutorial;

public class ResourceAnchor
{
    public ResourceAnchor()
    {
    }
}
```

HelloVenus.java

```
package vamp.tutorial;

import java.io.*;
import java.awt.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import javax.jnlp.*;

public class HelloVenus extends JFrame
{
 private WebBrowser _browser;

 public HelloVenus()
 {
   super( "Hello Venus - Java Web Start Resource Loading Tutorial" );

   addWindowListener( new WindowAdapter() {
     public void windowClosing( WindowEvent e ) { System.exit(0); }
   } );

   ClassLoader cl = ResourceAnchor.class.getClassLoader();

   ImageIcon appIcon = new ImageIcon(
     cl.getResource( "vamp/tutorial/images/inform.gif" ) );

   setIconImage( appIcon.getImage() );

   Properties props = new Properties();
   try
   {
     props.load(
       cl.getResourceAsStream( "vamp/tutorial/menu.properties" ));
   }
   catch( IOException ioex )
   {
     System.err.println( "*** failed to load properties: " + ioex.toString() );
   }

   _browser = new WebBrowser();

   StringBuffer html = new StringBuffer();

   html.append( "<html>" );
   html.append( "<h1>Java Web Start Resource Loading Tutorial</h1>" );

   // add properties to page
```

```
html.append( "<h2>Properties</h2>" );
html.append( "<table border='1'>" );
html.append( "<tr><th>Name<th>Value" );

// sort properties by putting them into a tree map
TreeMap propsSorted = new TreeMap( props );

Iterator it = propsSorted.entrySet().iterator();
while( it.hasNext() )
{
  Map.Entry entry = (Map.Entry) it.next();

  String key   = (String) entry.getKey();
  String value = (String) entry.getValue();

  html.append( "<tr><td>" + key + "<td>" + value );
}

html.append( "</table>" );
html.append( "</html>" );

_browser.displayHtmlText( html.toString() );

getContentPane().setLayout( new BorderLayout() );
getContentPane().add(
   new JScrollPane( _browser.getComponent() ), BorderLayout.CENTER );

BasicService basic = null;
try
{
  basic = (BasicService) ServiceManager.lookup( "javax.jnlp.BasicService" );
}
catch( UnavailableServiceException uex )
{
  System.err.println( "*** service unavailable; big trouble ahead: " + uex.toString() );
}

JMenuBar menuBar = new JMenuBar();

JMenu helpMenu = new JMenu( "Help" );

ImageIcon worldIcon = new ImageIcon(
   cl.getResource( "vamp/tutorial/images/world2.gif" ) );

String items = props.getProperty( "help.menu", "" );
StringTokenizer t = new StringTokenizer( items, " " );

while( t.hasMoreTokens() )
{
  String itemKey = t.nextToken();

  String itemText  = props.getProperty( itemKey+".text" );
  String itemUrl   = props.getProperty( itemKey+".url" );

  try
  {
    Action action = new CmdShowUrl( itemText, worldIcon, basic, new URL( itemUrl ) );
    helpMenu.add( action );
  }
  catch( MalformedURLException mex )
  {
    System.err.println( "*** invalid URL " + itemUrl +": " + mex.toString() );
  }
}

menuBar.add( helpMenu );
```

```
    setJMenuBar( menuBar );

    setSize( 600, 400 );
    setVisible( true );
  }
```

CmdShowUrl.java

```
package vamp.tutorial;

import java.awt.event.*;
import javax.swing.*;
import java.net.*;
import javax.jnlp.*;

public class CmdShowUrl extends AbstractAction
{
  private URL _url;
  private BasicService _basic;

  public CmdShowUrl( String text, Icon icon, BasicService basic, URL url )
  {
    super( text, icon );

    _basic = basic;
    _url = url;
  }

  public void actionPerformed( ActionEvent e )
  {
    _basic.showDocument( _url );
  }
}
```

WebBrowser.java

```
package vamp.tutorial;

import java.io.*;
import java.net.*;
import javax.swing.*;

import javax.swing.event.*;
import javax.swing.text.html.*;

public class WebBrowser
{
  private JEditorPane _browser;

  public WebBrowser()
  {
      _browser = new JEditorPane();
      _browser.setEditable( false );
      _browser.setContentType( "text/html" );
  }

  public void displayHtmlText( String html ) { _browser.setText( html ); }

  public JComponent getComponent() { return _browser; }
}
```

Rachel Resource Loading Example Code

CrossRefAnchor.java

```
public class CrossRefAnchor
{
    public CrossRefAnchor()
    {
    }
}
```

JavaDocAnchor.java

```
public class JavaDocAnchor
{
   public JavaDocAnchor()
   {
   }
}
```

ThemeAnchor.java

```
public class ThemeAnchor
{
   public ThemeAnchor()
   {
   }
}
```

HelloRachel.java

```
import com.l2fprod.gui.*;
import com.l2fprod.gui.plaf.skin.*;
import com.l2fprod.gui.plaf.skin.impl.gtk.*;
import com.l2fprod.gui.plaf.skin.impl.kde.*;

import java.io.*;
import java.awt.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;

import org.apache.log4j.*;

import vamp.html.*;
import vamp.http.*;
import vamp.http.loader.*;
import vamp.url.*;


public class HelloRachel extends JFrame
```

```
{
  static Category T = Category.getInstance( HelloRachel.class );

  private WebBrowser _browser;

  public HelloRachel()
  {
    super( "Hello Rachel - Java Web Start Resource Loading Example Application" );

    addWindowListener( new WindowAdapter() {
      public void windowClosing( WindowEvent e ) { System.exit(0); }
    } );

    // build URLs

    URL displayIconUrl = null;
    URL appIconUrl = null;

    URL javaDocUrl = null;
    URL javaDocUrl2 = null;
    URL crossRefUrl = null;
    URL crossRefUrl2 = null;

    try
    {
      displayIconUrl   = new URL( "class://ThemeAnchor/images/world2.gif" );
      appIconUrl       = new URL( "class://ThemeAnchor/images/inform.gif" );

      javaDocUrl   = new URL( "http://localhost:" + PORT + "/javadoc/index.html" );
      crossRefUrl  = new URL( "http://localhost:" + PORT + "/crossref/index.html" );

      javaDocUrl2  = new URL( "class://JavaDocAnchor/javadoc/index.html" );
      crossRefUrl2 = new URL( "class://CrossRefAnchor/crossref/index.html" );
    }
    catch( MalformedURLException mex )
    {
      T.error( "*** failed to create URL: " + mex.toString() );
    }

    ImageIcon appIcon =  new ImageIcon( appIconUrl );
    setIconImage( appIcon.getImage() );

    _browser = new WebBrowser();

    StringBuffer html = new StringBuffer();

    html.append( "<html>" );

    // -- add system properties to page

    html.append( "<h2>System Properties</h2>" );
    html.append( "<table border='1'>" );
    html.append( "<tr><th>Name<th>Value" );

    // sort system properties by putting them into a tree map
    TreeMap props = new TreeMap( System.getProperties() );

    Iterator it = props.entrySet().iterator();
    while( it.hasNext() )
    {
      Map.Entry entry = (Map.Entry) it.next();

      String key   = (String) entry.getKey();
      String value = (String) entry.getValue();

      html.append( "<tr><td>" + key + "<td>" + value );
```

```
}

html.append( "</table>" );

html.append( "<h2>Other</h2>" );

html.append( "ClassLoader hierachy" );
html.append( "<ul>" );
ClassLoader cl = getClass().getClassLoader();
while( cl != null )
{
  html.append( "<li>" + cl.getClass().getName() );
  cl = cl.getParent();
}
html.append( "</ul>" );

html.append( "SystemClassLoader hierachy" );
html.append( "<ul>" );
cl = ClassLoader.getSystemClassLoader();
while( cl != null )
{
  html.append( "<li>" + cl.getClass().getName() );
  cl = cl.getParent();
}
html.append( "</ul>" );

html.append( "<hr>" );
html.append( "SecurityManager: " );

SecurityManager sm = System.getSecurityManager();
if( sm != null )
  html.append( sm.getClass().getName() );

html.append( "</html>" );

_browser.displayHtmlText( html.toString() );

getContentPane().setLayout( new BorderLayout() );
getContentPane().add( new JScrollPane( _browser.getComponent() ), BorderLayout.CENTER )

// create menu
ImageIcon displayIcon =  new ImageIcon( displayIconUrl );

JMenuBar menuBar = new JMenuBar();

JMenu favoritesMenu = new JMenu( "Favorites" );

favoritesMenu.add(
  new DisplayUrlAction( "Java Doc @ " + javaDocUrl.toExternalForm(),
        displayIcon, _browser, javaDocUrl ) );

favoritesMenu.add(
  new DisplayUrlAction( "Java Doc @ " + javaDocUrl2.toExternalForm(),
        displayIcon, _browser, javaDocUrl2 ) );

favoritesMenu.add(
  new DisplayUrlAction( "Cross Reference @ " + crossRefUrl.toExternalForm(),
        displayIcon, _browser, crossRefUrl ) );

favoritesMenu.add(
  new DisplayUrlAction( "Cross Reference @ " + crossRefUrl2.toExternalForm(),
        displayIcon, _browser, crossRefUrl2 ) );

menuBar.add( favoritesMenu );
setJMenuBar( menuBar );

setSize( 600, 400 );
```

```
    setVisible( true );
  }

  public final static int PORT = 6868;

  public static void main( String args[] )
  {
    BasicConfigurator.configure();

    // enable class:// urls
    URL.setURLStreamHandlerFactory( new VampUrlFactory() );

    // set look&feel,theme,skin etc.
    try
    {
      URL themepack = new URL( "class://ThemeAnchor/skinlf/themes/modern.zip" );

      Skin skin = SkinLookAndFeel.loadThemePack( themepack );
      SkinLookAndFeel.setSkin(skin);
      SkinLookAndFeel lnf = new SkinLookAndFeel();
      UIManager.setLookAndFeel(lnf);
      UIManager.getLookAndFeelDefaults().put("ClassLoader",
                  lnf.getClass().getClassLoader());
    }
    catch( Exception ex )
    {
      T.error( "*** failed to change look&feel: " + ex.toString() );
    }

    // register resource anchors

    WebResourceManager roots = WebResourceManager.getInstance();
    roots.addResourceLoader( new ClassResourceLoader( CrossRefAnchor.class ) );
    roots.addResourceLoader( new ClassResourceLoader( JavaDocAnchor.class ) );

    try
    {
      // start web server
      WebServer http = new WebServer( PORT, roots );
      http.start();
    }
    catch( IOException ioex )
    {
      T.error( "*** failed to start web service: " + ioex.toString() );
    }

    HelloRachel gui = new HelloRachel();
  }
}
```

DisplayUrlAction.java

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

import vamp.html.*;

public class DisplayUrlAction extends AbstractAction
{
  private WebBrowser _browser;
  private URL _url;

  public DisplayUrlAction( String text, Icon icon, WebBrowser browser, URL url )
```

```
  {
    super( text, icon );

    _browser = browser;
    _url = url;
  }

  public void actionPerformed( ActionEvent e )
  {
    _browser.displayUrl( _url );
  }
}
```