Cincom

VisualWorks®

Plugin Developer's Guide

P46-0130-04

Part Number: P46-0130-04

Software Release 7.3

Cincom Systems, Inc.

55 Merchant Street

Cincinnati, Ohio 45246

Phone: (513) 612-2300

Fax: (513) 612-2000

World Wide Web: http://www.cincom.com

Contents

About This Book

This manual describes the VisualWorks® Plugin Development Environment (PDE) and how to use it to develop an application for deployment on a web page.

Audience

This book assumes you are familiar with Smalltalk and VisualWorks. Little knowledge of programming for the World Wide Web is required, though some applications will require extensive knowledge of this. Refer to the VisualWorks manuals for more information about VisualWorks programming. A large number of books and tutorials are available from commercial book sellers on programming for the web.

Conventions

We have followed a variety of conventions, which are standard in the VisualWorks documentation.

Typographic Conventions

The following fonts are used to indicate special terms:

Example	Description
template	Indicates new terms where they are defined, emphasized words, book titles, and words as words.
`cover.doc`	Indicates filenames, pathnames, commands, and other constructs to be entered outside VisualWorks (for example, at a command line).
`filename.xwd`	Indicates a variable element for which you must substitute a value.
windowSpec	Indicates Smalltalk constructs; it also indicates any other information that you enter through the VisualWorks graphical user interface.

Example	Description
Edit menu	Indicates VisualWorks user-interface labels for menu names, dialog-box fields, and buttons; it also indicates emphasis in Smalltalk code samples.

Special Symbols

This book uses the following symbols to designate certain items or relationships:

Examples	Description
File → New	Indicates the name of an item (New) on a menu (File).
<Return> key <Select> button <Operate> menu	Indicates the name of a keyboard key or mouse button; it also indicates the pop-up menu that is displayed by pressing the mouse button of the same name.
<Control>-<g>	Indicates two keys that must be pressed simultaneously.
<Escape> <c>	Indicates two keys that must be pressed sequentially.
Integer>>asCharacter	Indicates an instance method defined in a class.
Float class>>pi	Indicates a class method defined in a class.

Mouse Buttons and Menus

VisualWorks supports a one-, two-, or three-button mouse common on various platforms. Smalltalk traditionally expects a three-button mouse, where the buttons are denoted by the logical names <Select>, <Operate>, and <Window>:

<Select> button	*Select* (or choose) a window location or a menu item, position the text cursor, or highlight text.
<Operate> button	Bring up a menu of *operations* that are appropriate for the current view or selection. The menu that is displayed is referred to as the *<Operate> menu*.
<Window> button	Bring up the menu of actions that can be performed on any VisualWorks *window* (except dialogs), such as **move** and **close**. The menu that is displayed is referred to as the *<Window> menu*.

These buttons correspond to the following mouse buttons or combinations:

	3-Button	2-Button	1-Button
<Select>	Left button	Left button	Button
<Operate>	Right button	Right button	<Option>+<Select>
<Window>	Middle button	<Ctrl> + <Select>	<Command>+<Select>

Note: This is a different arrangement from how VisualWorks used the middle and right buttons prior to 5i.2.
If you want the old arrangement, toggle the **Swap Middle and Right Button** checkbox on the **UI Feel** page of the Settings Tool.

Getting Help

There are many sources of technical help available to users of VisualWorks. Cincom technical support options are available to users who have purchased a commercial license. Public support options are available to both commercial and non-commercial license holders.

Commercial Licensees

If, after reading the documentation, you find that you need additional help, you can contact Cincom Technical Support. Cincom provides all customers with help on product installation. For other problems there are several service plans available. For more information, send email to supportweb@cincom.com.

Before Contacting Technical Support

When you need to contact a technical support representative, please be prepared to provide the following information:

- The *version id,* which indicates the version of the product you are using. Choose **Help → About VisualWorks** in the VisualWorks main window. The version number can be found in the resulting dialog under **Version Id:**.
- Any modifications (*patch files*) distributed by Cincom that you have imported into the standard image. Choose **Help → About VisualWorks** in the VisualWorks main window. All installed patches can be found in the resulting dialog under **Patches:**.

- The complete error message and stack trace, if an error notifier is the symptom of the problem. To do so, select **copy stack** in the error notifier window (or in the stack view of the spawned Debugger). Then paste the text into a file that you can send to technical support.

Contacting Technical Support

Cincom Technical Support provides assistance by:

Electronic Mail

To get technical assistance on VisualWorks products, send email to supportweb@cincom.com.

Web

In addition to product and company information, technical support information is available on the Cincom website:

http://supportweb.cincom.com

Telephone

Within North America, you can call Cincom Technical Support at (800) 727-3525. Operating hours are Monday through Friday from 8:30 a.m. to 5:00 p.m., Eastern time.

Outside North America, you must contact the local authorized reseller of Cincom products to find out the telephone numbers and hours for technical support.

Non-Commercial Licensees

VisualWorks Non-Commercial is provided “as is,” without any technical support from Cincom. There are, however, on-line sources of help available on VisualWorks and its add-on components. Be assured, you are *not* alone. Many of these resources are valuable to commercial licensees as well.

The University of Illinois at Urbana-Champaign very kindly provides several resources on VisualWorks and Smalltalk:

- A mailing list for users of VisualWorks Non-Commercial, which serves a growing community of VisualWorks Non-Commercial users. To subscribe or unsubscribe, send a message to:

  vwnc-request@cs.uiuc.edu

  with the SUBJECT of "subscribe" or "unsubscribe".

- An excellent Smalltalk archive is maintained by faculty and students at UIUC, who are long-time Smalltalk users and leading lights in the Smalltalk community, at:

http://st-www.cs.uiuc.edu/

- A Wiki (a user-editable web site) for discussing any and all things VisualWorks related at:

  http://wiki.cs.uiuc.edu/VisualWorks

- A variety of tutorials and other materials specifically on VisualWorks at:

  http://wiki.cs.uiuc.edu/VisualWorks/Tutorials+and+courses

The Usenet Smalltalk news group, comp.lang.smalltalk, carries on active discussions about Smalltalk and VisualWorks, and is a good source for advice.

Additional Sources of Information

This is but one manual in the VisualWorks library. The Cincom Smalltalk publications website:

http://www.cincom.com/smalltalk/documentation

is a resource for the most up to date versions of VisualWorks manuals and additional information pertaining to Cincom Smalltalk.

Online Help

When you install the PDE, a help file is added to the VisualWorks online help system. If you already have help installed in your image, select **Help → Rebuild Help Library** to add this help to the system. Otherwise, it is included automatically when you load help.

1

The VisualWorks Plugin

Introduction

VisualWorks® has always provided a cross-platform application environment, allowing you to develop an application on one platform and deploy it, unchanged, on any other of its supported platforms. This kind of portability makes VisualWorks an ideal environment for developing web-based applications, or applets. Using a web-browser as the GUI display system, the VisualWorks Plugin greatly simplifies the deployment of many applications.

ActiveX Control

The VisualWorks Plugin for VisualWorks is an ActiveX Control which can be installed in Internet Explorer (or other fully ActiveX capable browser) to allow a VisualWorks application to operate inside the web browser window as an embeded object.

> **Note:** The VisualWorks 7.3 release provides only the ActiveX Control version of the plugin. A future release will also include a version of the plugin which implements the new cross-browser NPAPI and will run in newer versions of NPAPI enabled browsers such as Netscape, Mozilla, Opera and Firefox.

Developing VisualWorks Applets

The VisualWorks Plugin works in a way that makes developing an application or applet targeted for deployment in a browser no different from developing an application for stand-alone deployment. This is possible because of how the plugin connects the browser and the VisualWorks runtime environment.

The plugin establishes a parent-child relation between the browser window and the VisualWorks application window, such that when the application draws to the VisualWorks window, the drawing instructions are passed through to the browser window. This makes programming an applet as simple as programming a normal VisualWorks application.

The plugin itself is very small, handling the interface between the VisualWorks virtual machine (VM) and the web browser to run a VisualWorks application UI within a browser window. Additional browser API in VisualWorks further enables the applet to perform specific browser operations, such as setting the status bar and performing HTTP GET and POST operations. The applet itself consists of an image installed with the plugin VM, and a set of installed or downloadable parcels.

Deploying an applet is as simple as referencing it in an OBJECT tag in an HTML document. The tag specifies the location of a versioned Internet Component Download CAB deployment file containing the application. When the page is opened, this file is downloaded and the control is installed, the VM is launched and the parcel is loaded into the running image. For large applications, and especially for intranet applications, you may find it more suitable to build a custom image containing the bulk of the application code. The downloadable parcels then contain only specific code for the application.

Using a VisualWorks Applet

The end-user experience of using a VisualWorks Plugin applet is nearly transparent.

When a web page containing a VisualWorks applet is opened, VisualWorks starts, the applet parcel is loaded (along with any specified prerequisite parcels), and then the applet is launched. Because the applet parcel is typically small, and VisualWorks loads very quickly, the user experiences only a small delay before the applet is ready for use.

If the VisualWorks Plugin is not already installed when the page is opened, the plugin is downloaded and installed in accordance with the security settings of the user's browser.

When the user leaves the page containing the applet, the applet closes. When all pages running applets close, the VisualWorks image itself closes. If the user goes back to a page with a closed applet, the applet relaunches.

System Requirements

The VisualWorks Plugin development environment is a standard VisualWorks 7.3 development image.

Applets developed for the VisualWorks Plugin can be viewed in the following:

- Microsoft Internet Explorer 5.0 or higher

The VisualWorks Plugin ActiveX Control requires Unicode support and therefore is supported on:

- Microsoft Windows NT 4.0, or later

If your plugin applet must support a browser client on a platform which does not provide Unicode support (Windows 95/Me/98), please consult the Microsoft documentation concerning the Microsoft Layer for Unicode, and the instructions for including this translation layer in an ActiveX Control.

Compatibility

The VisualWorks Plugin ActiveX Control works with VisualWorks 7.3 or later.

The Plugin parcel for VisualWorks 7.3 and later is not backward compatible with the old Plugin and PluginBase parcels. The PluginBase parcel is obsolete. If you have an existing custom Plugin image you must rebuild the image with the new parcel. You must also compile and test your application parcel. The plugin API internal to the AppletModel class has changed very little, so it is likely your application will not need any modification unless you make significant use of GET and/or POST, especially POST from a file, which is no longer supported.

Deployment of an ActiveX Control uses Internet Component Download to install and register the control on the user's machine. This process is significantly different from the installation procedures for the previous VisualWorks Plugin. Refer to Chapter 6, "Packaging Your Custom Plugin" for information on deploying your applet.

2

Getting Started

The Plugin Components

The main components of the VisualWorks Plugin are:

- The plugin file (a DLL on Windows platforms) that connects VisualWorks and the web browser
- An external, translatable message resource file
- A generic, platform independent, base image
- A platform specific virtural machine, which is an executable file

To this set of files, you add one or more parcel files containing your application, and possibly a custom image file to run in place of the generic image.

Running the VisualWorks Plugin Examples

Note: The generic VisualWorks Plugin is not a signed control, and because any Smalltalk parcel may be loaded into its image, it cannot be marked safe for either initialization or scripting. You must set your browser, at minimum, to prompt you to load an unsigned control and to initialize and run an unsafe control.

Note: If you have very strict security configured, you will not be able to download and install the Plugin. If you are using a web server on your local machine, you can change the security settings for the Local Intranet Zone only.

To install the generic VisualWorks Plugin ActiveX Control and run the

various plugin examples, first create the web server location which will serve the VisualWorks Plugin CAB deployment file through Internet Component Download.

1. Create a virtual directory named **`plugin-install/`** on your web server. The web server can be either local or remote, and it can be IIS, Apache or any other web server you desire.
2. Copy **`plugin/deploy/vwpluginax.cab`** into the physical directory referenced by your **`plugin-install`** virtual directory.
3. Open **`plugin/examples/index.html`** in Internet Explorer and follow the instructions on that page to access the plugin examples.

The VisualWorks Plugin ActiveX Control will be downloaded and installed the first time you access an HTML page that embeds a VisualWorks Plugin applet. You will be prompted to install the unsigned control only once. As you run additional pages containing VisualWorks Plugin applets you will be prompted about the unsafe control each time the page opens.

When you deploy your own VisualWorks Plugin you will need to consider signing the control, and you must ensure that your control is truly safe before you mark it safe for initialization and/or scripting. Please see Chapter 6, “Packaging Your Custom Plugin” for deployment considerations.

The Plugin Development Environment

The plugin development environment is a standard VisualWorks development image into which you load the PluginDev parcel and its prerequisite, the Plugin parcel. These parcels consist of a set of classes and tools that simplify developing a VisualWorks applet. Extensions provide the ability to:

- generate parcels for loading the applet
- generate a custom plugin image for a complete application
- execute a debug image, for normal VisualWorks debugging
- package a custom application image for web delivery

A subclass of ApplicationModel, called AppletModel, invokes plugin API functions for directly accessing the browser. For applets, create your application as a subclass of AppletModel.

Loading Plugin Development Support

Installation instructions for VisualWorks are provided in the VisualWorks *Installation Guide*. Once VisualWorks is installed, the plugin development environment must be loaded into the development image. The necessary features are all contained in the Plugin and PluginDev parcels.

To load the development environment, open the Parcel Manager and select the **Directories** page. In the Plugin directory select the PluginDev parcel, then pick **Parcel Load**. Both parcels then load (Plugin is a prerequisite of PluginDev).

At this point, you are ready to begin developing an applet.

3

Developing a Plugin Applet

The process of developing a plugin applet is generally the same as developing any VisualWorks application, but there are a few differences in detail.

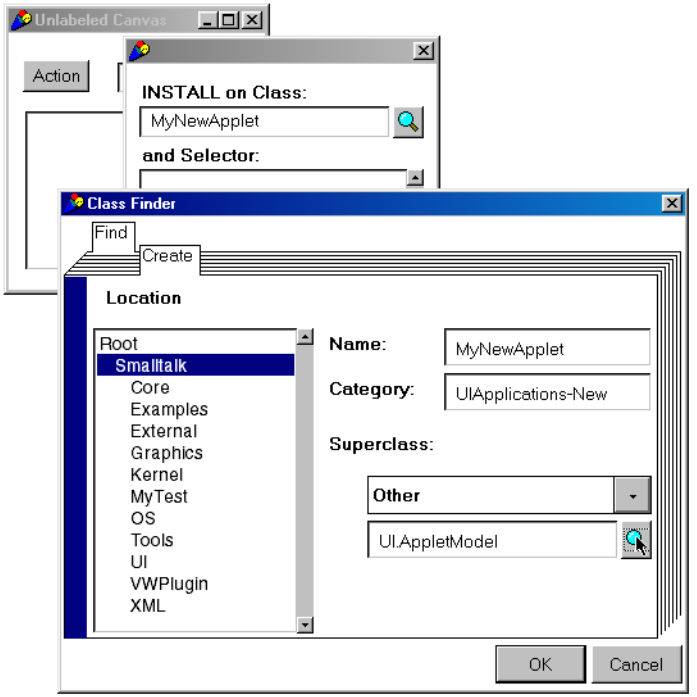
Subclassing AppletModel

The VisualWorks Plugin parcel adds a subclass of ApplicationModel, called AppletModel, which adds protocol specifically for interactions between VisualWorks and the web browser plugin API.

To create your application as a subclass of AppletModel, simply specify AppletModel as the superclass when installing a new canvas:

1. In the Canvas Tool, click **Install**.
2. In the Install On Class dialog, enter a name for the class and click **OK**.
3. In the Class Finder dialog, select **Other** for the superclass, then click the browser (magnifying glass) button.
4. In the new Class Finder dialog, select the UI name space, then find and select the AppletModel class. Click **OK** to close that dialog and continue.
5. Back in the first Class Finder dialog, select a name space for your application and click **OK**.

   The name space can be one of your own or an existing name space, such as UI, that will be present in the distribution image. If it is your own, make sure it is defined by the applet parcel by adding its definition to that parcel.



For an existing subclass of ApplicationModel, you can either move it to AppletModel by changing its superclass, or adapt it to include some of the plugin API, as described in Adapting an Existing Application below.

AppletModel handles opening your application in the web browser, just as ApplicationModel opens a normal VisualWorks application. When launching an applet in a browser, AppletModel responds to the VWOPEN attribute in the OBJECT HTML tag.

Since AppletModel also responds to the standard VisualWorks opening protocol, you can open an instance of your application as usual in the development environment. For early development, this is adequate.

Running an Applet in a Browser

To access browser capabilities, such as to perform HTTP GET or POST operations, the applet needs to run in the web browser context. This requires creating a sample HTML page containing an OBJECT tag specifying the applet attributes.

For the full set of attributes supported in the OBJECT tag, refer to Chapter 4, “Deploying an Applet.”

The **`plugin/examples`** directory contains a number of HTML pages which can be used as templates. For testing purposes you only need a simple page with a subset of the attributes. The source for your HTML document might be simply:

```
<HTML>
<HEAD> </HEAD>
<BODY>
<P>This is just a test.</P>
<OBJECT  CLASSID="CLSID:FF48278C-094A-4188-95AA-4B1E03F3163C"
    CODEBASE=
        "http://mydomain/plugin-install/vwpluginax.cab#version=2,0,X,YYY"
    WIDTH="500" HEIGHT="300">
        <PARAM NAME="PARCEL" VALUE="MyAppletParcel.pcl">
        <PARAM NAME="VWOPEN" VALUE="MyApplet">
    </OBJECT>
</BODY>
</HTML>
```

The CLASSID attribute contains the unique GUID identifier for the generic VisualWorks Plugin ActiveX Control. It must be specified exactly as shown.

The CODEBASE attribute references the URL of the versioned CAB deployment file used to download and install the ActiveX Control. Please consult the VisualWorks Plugin example file **`hello-x.html`** for the version number of the deployment file distributed with this release. If you have already installed the generic VisualWorks Plugin by opening a page containing a VisualWorks Plugin applet, the browser will not install it again unless the version referenced in the CODEBASE attribute references a newer version of the CAB deployment file.

The WIDTH and HEIGHT attributes also need to be set for your applet.

The PARAM attributes provide parameters used by the VisualWorks Plugin. The PARCEL parameter specifies the parcel containing the applet, MyAppletParcel.pcl in the example. The VWOPEN parameter specifies the name of the AppletModel subclass that defines the applet, MyApplet in the example.

Save this HTML file in the same directory with your application parcel files.

Using the Plugin Debugger

Because you can open an AppletModel application in the development image just like a normal ApplicationModel application, you can do a great deal of testing and debugging the same way you do with other VisualWorks applications. However, when you start developing aspects that depend on web browser capabilities, you need to run the applet in the browser.

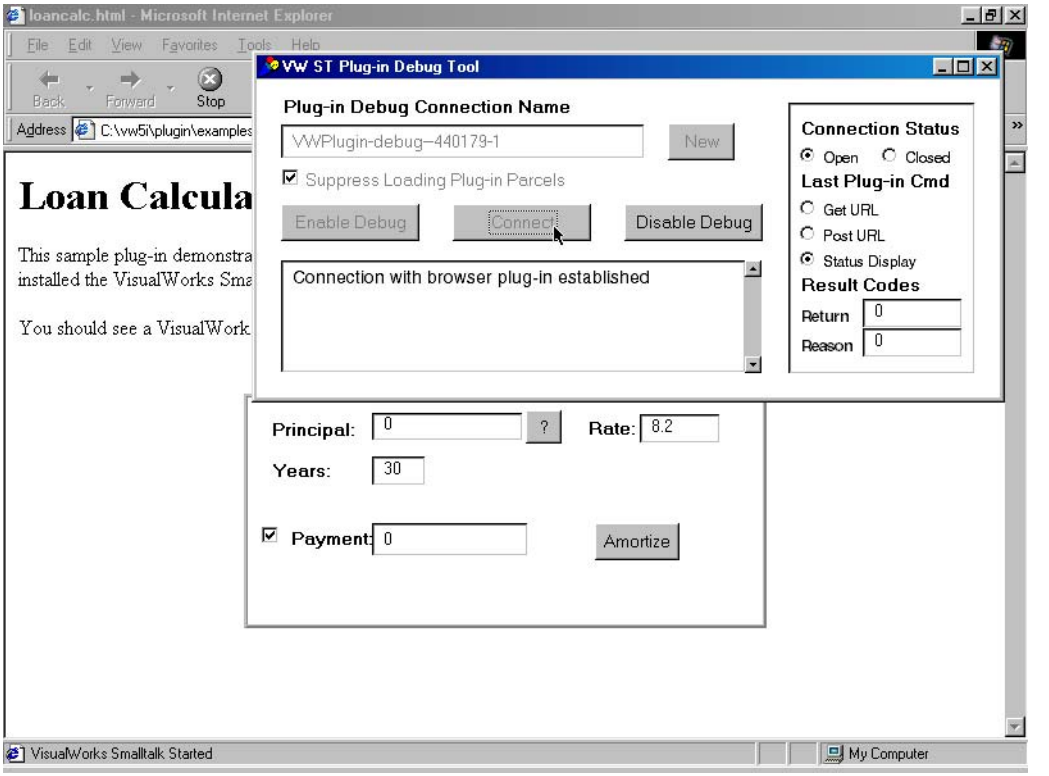
The plugin Debug Tool allows you to connect to an applet running in a web browser and do interactive debugging on it, using the development environment. The applet runs in the development image, giving you full access to development features. You also have the option to run the debugger either on the deployed parcel, or on the development parcel loaded in the development image.

Starting a Plugin Debug Session

Before launching the applet page, open the Debug Tool by selecting **Tools → Plugin Debug Tool** in the VisualWorks Launcher. The tool displays instructions for launching a session in its text panes. In short, the procedure is:

1. Start the plugin Debug Tool.
2. Click **Enable Debug**.
3. Open the HTML page containing the applet in a web browser.
4. Click **Connect**

.



The **Suppress Loading Plugin Parcels** checkbox determines whether the deployed parcels are loaded when the applet is launched:

- When checked, parcels are not loaded, so the application parcels need to be loaded before launching the applet. For debugging in a development session, it is normal to pre-load the development parcels so source code is available.
- When unchecked, the image attempts to load the deployed version of the parcel. If the development version is already loaded, you are notified.

When you click **Connect**, the development image is connected to the applet running in the browser. You can now debug and edit the applet's state as usual, because it is running in the development image.

When you are finished with a debug session, click **Disable Debug**. This resets the debug registry entry, so your applets can start in normal runtime mode.

Note: If you forget to disable debugging before you shut down your VisualWorks image, the VisualWorks Plugin will not run unless you have a VisualWorks image open. To restore the debug registry entry to its normal runtime state, open the Debug Tool, click **Connect** and then click **Disable Debug**.

What You Can Do in a Debug Session

The plugin Debug Tool gives you the power of traditional Smalltalk interactive development, but on a live applet. Among other things, you can:

- Press Ctrl-Y in the browser to open a User Interrupt debug window.
- Enter self halt in code to cause breaks in the processing.
- Open inspectors to examine the applet state.
- Change the applet's state or methods, to change its behavior while running.

In short, any trick you might play in debugging a VisualWorks application during development is now available on the applet.

Adapting an Existing Application

If you have an existing application that you want to deploy as an applet, but do not want to change its subclass to AppletModel, you can do so by duplicating a subset of the AppletModel protocol in your application class.

The protocol can be added to your class by filing in **`applet-api.st`** from the **`plugin`** subdirectory. The essential protocol is:

isAppletModel
Must return true for an instance functioning as an applet model and open using the VWOPEN attribute.

pluginConnection: *anInternetBrowserConnection*
Set the connection that is used for communicating with the browser. If only one connection is used (all that are presently supported), this method does not need to actually do anything (see pluginConnection).

pluginConnection

Answer the connection with the plugIn for this instance. Since only one plugIn connection is supported the following can be used instead of saving the value provided in pluginConnection:

```
^VWPlugin.InternetBrowserConnection current
```

postBuildWith: *aBuilder*

Set the damageRepairPolicy to use double-buffering to help reduce flickering in the plugin.

If your applet opens a VisualWorks child window, and you want to return focus to the main applet when that window closes (see the Loan Calculator example), you will also need the following:

pluginConnectionIsOpen

Answer whether the plugin connection is established.

activatePluginWindow

Send a message to the VisualWorks Plugin to return focus to the main plugin window on the HTML page.

4

Deploying an Applet

When a web page containing a VisualWorks applet is opened, VisualWorks starts, the applet parcel is loaded (along with any specified prerequisite parcels), and then the applet is launched. If the VisualWorks Plugin is not already installed when the page is opened, the plugin is downloaded and installed in accordance with the security settings of the user's browser.

The standard plugin image (**`plugin-base.im`**) contains the runtime support classes and protocol required for a simple applet. This is the image that is installed with the generic VisualWorks Plugin. To run an applet in this image, all applet code must be contained in the parcel or parcels that download from the web site. Since applets tend to be small, there is usually only a single, small parcel, and downloading is very quick.

For large applications, you may need to build a custom image that will be installed as a custom instance of the plugIn. This is described in the next chapter, Chapter 5, "Building a Custom Plugin Image."

The **`plugin\examples`** directory contains a set of VisualWorks applets and HTML pages that you may examine as examples for constructing your applet.

Parceling the Applet

VisualWorks provides several paths to defining and creating parcels. For example, you can use parcels as your source code storage format, in which case you define them at the beginning of a project. Or, you may use the VisualWorks Store Repository for code storage, and only define parcels at the end of the process.

A deployed plugin application does not generally include source code, but you do not have to save the parcel any differently for deployment than you do for development. Save the parcel as normal, but then only upload the **`.pcl`** file to the web server. When uploading parcels to the web server, treat the parcel files as binary files.

While there are alternatives for adding definitions to parcels, here's a standard approach:

1. In a Parcel Browser (or parcel view of the System Browser), select **Parcel → New…**, and enter a name for the parcel.

   The name may be the same as the name for the actual parcel file, but need not be. *Do not* include a filename extension.

2. In the browser, find each class used in defining your application, and select **Class → Parcel → Move to…**, and select the parcel.

3. If your application extends classes in the target image, find each "loose method" in the browser, and select **Method → Parcel → Move to…**, and select the parcel.

   Note that if you have put all your loose methods for a class in a special protocol, you can add the protocol to the parcel instead, and move all of the methods in a single operation.

4. Select the parcel, and select **Parcel → Save…** .

5. In the parcel saving dialog, enter the file name for the parcel (without the extension, which will be provided), then click **OK**.

Adding an Application to a Web Page

You add your application to a web page using the HTML OBJECT tag. The standard HTML options are supported, plus a a number of tag parameter attributes specifically for the VisualWorks Plugin.

For example:

```
<OBJECT
    CLASSID="CLSID:FF48278C-094A-4188-95AA-4B1E03F3163C"
    TYPE="application/x-visualworks-parcel"
    CODEBASE=
        "http://mydomain/plugin-install/vwpluginax.cab#version=2,0,X,YYY"
    WIDTH="500" HEIGHT="300">
        <PARAM NAME="PARCEL" VALUE="MyApplet.pcl">
        <PARAM NAME="VWOPEN" VALUE="MyApplet">
        <B>The ActiveX PlugIn was not installed!</B>
    </OBJECT>
```

The relevant tag attributes are:

CLASSID
The unique GUID identifier of the ActiveX Control.

CODEBASE
The URL which can be used to download and install the plugin if it is not already installed on the user's machine. Please consult the VisualWorks Plugin **`readme.txt`** file for the current version of the deployment file for this release.

WIDTH
The width of the applet window.

HEIGHT
The height of the applet window.

TYPE
The MIME type for the plugIn applet (currently application/x-visualworks-parcel). This is optional for the ActiveX Control.

The supported tag PARAM values are:

PARCEL
The parcel file containing the applet code. This can be an absolute URL or a file name relative to the location of the page containing the applet.

VWOPEN
Name of the class, typically a subclass of AppletModel, containing the window specification to be opened.

VWCODEBASE
URL for the location of additional parcels. Unless specified, the location is inferred to be the same as that for the PARCEL.

VWPRELOAD

A list of parcel file names, separated by commas, to be loaded into the image in the specified order before the main PARCEL. Unless specified as an absolute URL, the location of a preloaded parcel is inferred to be the VWCODEBASE.

VWAPPL

The application name associating this instance of the plugin with a specific image.

For the PARCEL or VWCODEBASE tag parameter attributes, you may also use the special tag $(VWPLUGIN) which indicates the file is relative to the VisualWorks Plugin installation directory on the local machine. This allows you to load a parcel from either a remote URL (subject to the standard ALLOW/DENY configuration criteria) or a parcel file in the Plugin installation directory (subject to ALLOW/DENY LOCAL configuration criteria). Compare **`example.html`** (in **`plugin\deploy`**) and **`hello-x.html`** (in **`plugin\examples`**) for the two formats.

The VisualWorks Plugin knows what version of the Plugin parcel (in the plugin base image) it can talk to. The DLL and the VisualWorks image must agree on the version before the image will start correctly. There is no option to specify a version of VisualWorks.

Any text following the PARAM attributes, before the close of the OBJECT tag, will be displayed in the HTML page if the plugin fails to install correctly.

The VWOPEN attribute in the OBJECT tag specifies the application to open:

```
<OBJECT …
    <PARAM NAME="VWOPEN" VALUE =”MyApplet”>
```

In this tag, MyApplet is the name of the subclass of AppletModel in which your applet is defined. AppletModel includes protocol specifically for communicating with the web browser window, and so should be used in most cases.

In cases where it is not practical to subclass AppletModel, specific protocol can be added to the class to handle essential browser functions. Refer to Chapter 5, “Building a Custom Plugin Image” for the specific protocol.

Testing the Deployed Applet

Testing the applet in a debugger session with development parcels preloaded is powerful, but cannot ensure that parcels load properly in a deployment environment. For final testing, you must allow the applet to start up using the deployed parcels.

As long as debugging is disabled (click **Disable Debug**), VisualWorks will load the deployment parcels as specified on the OBJECT tag. If the parcel source file is present, it is loaded as well.

You can also run a debug session with the deployment parcels loaded, and so debug a parcel loading problem. To do this:

1. Unload all parcels defining the applet from your development image.
2. Start the Plugin Debug Tool.
3. Uncheck **Suppress Loading Plugin Parcels**.
4. Click **Enable Debug**.
5. Open the web browser on the page containing the applet.
6. Click **Connect**.

As the plugin is launched, it loads the deployed parcels and any dependencies, as specified by the OBJECT tag. You can now debug the applet as it is currently deployed.

5

Building a Custom Plugin Image

The standard plugin image (**`plugin-base.im`**) contains the runtime support classes and protocol required for a simple applet. This is the image that is installed with the generic VisualWorks Plugin.

For a large application, as is more common for applications that are run over an intranet, downloading the entire application as one or more parcels each time it is run is inefficient. In this situation it is better to include much of the application in your CAB deployment file, in either:

- a custom image, or
- as additional parcels (invoked by the VWPRELOAD attribute in the OBJECT tag) that add functionality to the standard image.

In either case, the image or parcels only need to be downloaded once to each target machine. The applet parcel that is downloaded can then be quite small, perhaps containing only the GUI, defined in a subclass of AppletModel, that displays in the browser window.

In this chapter we describe building a custom instance of the plugin, based on a custom image.

Building a Custom Image

The standard plugin image (**`plugin-base.im`**) is completely generic, providing a run-time environment for an application that is completely contained in downloadable parcels. Such an application is fine until the size of the parcels becomes large enough to be inconvenient to download at each execution of the applet. For a large application, is is useful to create a custom base image that contains a significant part of the application, leaving only a small part for downloading, such as the GUI and associated logic.

The Runtime Packager, together with a special parameters file (**`vwplugin.rtp`**), prepares the development image for deployment as a plugin image.

The Runtime Packager "strips" the image, removing development support facilities such as development tools. Because it is a custom image, only parcels developed specifically for it should be loaded.

To create the custom plugin image:

1. Start the original **`visual.im`** supplied with VisualWorks.
2. Load the following parcels into the image:
   - RuntimePackager
   - Plugin (or PluginDev, which includes tools that will be removed during stripping)
3. Load the part of your application code that will be permanent in the deployed image. This is any of the application code that is not to be downloaded when the applet is launched.
4. (Optional) Unload parcel XML-Source and any other development parcels.

   Refer to the VisualWorks License Agreement for parcels that may not be included in a runtime image.
5. In the Settings Tool on the **Look and Feel** page, set **Window placement** to **Automatic**.

   This change, while strictly optional, makes any auxilliary application window open without the user placing and sizing the window. This is a more familiar behavior for most users.
6. Select **Tools → Runtime Packager**.
7. In Runtime Packager, select **File → Load Params**, and load **`plugin\vwplugin.rtp`**, the plugin Runtime Packager data file.
8. In Runtime Packager, select **Actions → Set common options**, and change the **Runtime Image Path Name** to the directory path and file name (without the .im suffix) for the deployed image, and click **OK**.

   You will use this image name in the plugin initialization file when identifying the base image for your application. (See Chapter 4, "Deploying an Applet" for more information.)
9. In Runtime Packager, select **Actions → Strip system**.

10 When the stripping process completes, launch the created image two more times as directed by the text windows. After the second launch, the text window should indicate that the image is ready for use.

Plugin Initialization File

To provide additional configuration information in a cross-platform format, the VisualWorks Plugin uses a plain text initialization file, **`vwplugin.ini`**. This file is installed in the VisualWorks Plugin installation directory.

The initialization file allows specific attributes to be set for individual instances of the plugin, as well as general attributes. When installing a new instance of the plugin, usually to run a custom image, you may also need to edit this file, either programmatically or manually, to adjust the attributes appropriately.

The initialization file uses the following basic attributes:

Initialization File Attributes

DIRECTORY	Names the “current directory” used for launching the VisualWorks Plugin image
OBJECTENGINE	Specifies the path to the executable virtual machine
BASEIMAGE	Specifies the path to the base image. This is generally needed for custom images.
APPLICATION	Specifies a value matching the VWAPPL tag attribute (see below)
EXTRA	Specifies extra information of interest to the application

There are also two security attributes, which are discussed in more detail below, under “Security Measures”:

Security Attributes

ALLOW	Specifies domain locations allowed to supply a plugin parcel
DENY	Specifies domain locations not allowed to supply a plugin parcel

Path names can be either absolute or relative file names (not URLs). Relative path names are relative to the value specified for DIRECTORY or to the VisualWorks Plugin installation directory if DIRECTORY is not supplied.

Note that the attribute names are not case-sensitive. However, any unknown attributes will cause an error dialog.

The initialization file can specify options for individual applications as well as setting global options. To allow this, begin a new section in the initialization file with the Application attribute, as in:

```
Application MyApp
```

The application name must match the name specified in the VWAPPL option in the OBJECT tag. Following this are attributes for this application only. For example, mixing general attributes and application specific attributes, the initialization file could look like this:

```
BaseImage plugin-win-base.im

Application MyApp
BaseImage myimage.im
Deny all
Allow myhost.com

Application Debug
Directory c:\vw7\image
ObjectEngine ..\bin\vwntdbg.exe
BaseImage mydebug.im
Deny all
Allow local
```

In this configuration, defaults are accepted for most attributes for the standard plugin image, but specific attributes are set for MyApp and for debugging.

Security Measures

The current security measures for the plugin employ a “trusted site” strategy, enforced by the ALLOW and DENY attributes specified in the plugin initialization file.

Identifying Trusted Sites

You identify trusted sites by defining a filtering scheme in the initialization file, using the ALLOW and DENY attributes. The attribute formats are:

```
ALLOW < all | local | hostID >
DENY < all | local | hostID >
```

Host IDs can be partial, and can be specified either as IP decimal values or as names.

IP decimal values specify from the network down, and unspecified levels match all values. For example, 216, 216.1, 216.1.2, and 216.1.2.3 all match 216.1.2.3.

Similarly, when specified by name, unspecified levels match all values. For example, com, cincom.com, and www.cincom.com all match www.cincom.com.

Allow and deny attributes can be mixed to provide the desired protection. For example, the following allows all cincom.com and advbool.com, except for test.advbool.com, and denies all other hosts except for local:

```
Deny all
Allow local
Allow cincom.com
Allow advbool.com
Deny test.advbool.com
```

How Sites are Verified

Every time a parcel is requested, the location of the request, whether specified by the PARCEL or VWCODEBASE parameters in the OBJECT tag, is compared to the trusted sites set by the ALLOW and DENY attributes. The parcel is downloaded, *only if* the site is allowed.

By default, if a site is not listed, it is allowed. To switch this to deny access to all sites not listed, begin the sequence with DENY ALL.

While the ALLOW and DENY parameters can be specified by decimal representations, they are checked only if the requested site is also specified using decimal representation. Translation from text to decimal is not done in the attempt to verify a site.

HTTP GET and POST requests are also tested for access permission before the command is issued, and only commands directed to an allowed site are actually issued.

6

Packaging Your Custom Plugin

For testing purposes, you can use the generic VisualWorks Plugin ActiveX Control. However, when you deploy your own application, you will need to address such issues as signing the control and marking your control safe for initialization and/or scripting, which are not possible using the generic plugin.

In this chapter we provide general instructions for packaging your applet which are applicable whether you are packaging a custom image for testing with the generic plugin control or packaging your own custom plugin control. We also discuss the process and considerations for the final step in deploying your VisualWorks Plugin ActiveX Control: building your own custom control.

Preparing to Test Your Application

To test a basic applet running in the standard VisualWorks Plugin image, there is no special packaging necessary. You can make the pre-packaged generic plugin deployment file, **`plugin\deploy\vwpluginax.cab`**, available for download and installation, and reference it and your applet parcel file in the OBJECT tag on a test HTML page.

If your application requires a custom image, on the other hand, you need to package it for installation with Internet Component Download by building your own CAB deployment file for your application.

Before you build your own deployment files to install your custom application, you should understand the general installation details for the VisualWorks Plugin ActiveX Control.

Installing the VisualWorks Plugin ActiveX Control

When the browser opens an HTML page which embeds the VisualWorks Plugin ActiveX Control for the first time, the versioned CAB deployment

file is downloaded from the URL referenced in the CODEBASE attribute of the OBJECT tag and installed through Internet Component Download in accordance with the security settings of the user's web browser. The CAB deployment file will normally not be downloaded again unless the CODEBASE attribute references a newer version of the control.

The VisualWorks Plugin DLL (**`axvwplugin.dll`**) is installed and registered in the ActiveX Cache Folder, (**`WINNT\Downloaded Program Files`** on Windows 2000). The remaining VisualWorks components are installed under the standard **`Program Files`** directory in **`VisualWorks\Plugin\2.00`**.

In addition to registering the ActiveX, the installation makes three entries in the Windows Registry :

HKLM\SOFTWARE\Cincom Systems,Inc.\VisualWorks\PlugIn\2.00\VWPluginDebug
HKLM\SOFTWARE\Cincom Systems,Inc.\VisualWorks\PlugIn\2.00\VWPluginDir
HKLM\SOFTWARE\Cincom Systems, Inc.\VisualWorks\PlugIn\2.00\VWPluginMsg

Removing the VisualWorks Plugin ActiveX Control

If you have installed the generic plugin you must first remove it from your machine before you can test your own application which uses a custom image. This is because the version of a CAB deployment file references the version of the ActiveX Control DLL it contains. As long as your CAB deployment file includes the generic VisualWorks Plugin ActiveX Control DLL, this version is already installed, and the new file, containing your custom image, will not be downloaded.

To uninstall the VisualWorks Plugin ActiveX Control,

1. Delete the entire Plugin installation folder **`VisualWorks\Plugin\2.00`** from the **`Program Files`** directory.
2. Remove the VWPlugin ActiveX Control from the ActiveX Cache Folder, **`WINNT\Downloaded Program Files`**.

The following registry items can optionally be removed, but this is not required:

HKEY_CLASSES_ROOT\AppID\{61ADB2B2-BF94-43B4-B04B-379B3E59BA67}
HKEY_CLASSES_ROOT\AppID\VWPlugin-AX
HKEY_CLASSES_ROOT\CLSID\{FF48278C-094A-4188-95AA-4B1E03F3163C}
HKEY_CLASSES_ROOT\VWPluginAX.VisualWorksAX
HKEY_CLASSES_ROOT\Interface\{19A5484E-1774-48DB-A119-840F8A73B5EB}
HKEY_CLASSES_ROOT\TypeLib\{61ADB2B2-BF94-43B4-B04B-379B3E59BA67}
HKEY_LOCAL_MACHINE\SOFTWARE\Cincom Systems, Inc.\VisualWorks\PlugIn\2.00

In most cases there is minimal benefit to removing these variables as the registry key is used only by VisualWorks.

Building Deployment Files For Your Application

To deploy your own application, you will need to build your own CAB deployment file. This requires the **`CABARC.EXE`** utility which is part of the Microsoft Cabinet Software Development Kit, and can be downloaded from the Microsoft support site. You must install the Microsoft Cabinet SDK before you can build your own deployment files.

To build your own CAB deployment file, copy the following files from the **`plugin\deploy`** directory to a work directory.

Deployment Directory Contents

File name	Description
`axvwm200.dll`	VisualWorks Plugin messages resource
`axvwplugin.dll`	VisualWorks Plugin ActiveX Control
`mkcabpluginax.bat`	Command file to generate deployment file
`plugin-base.im`	VisualWorks Plugin image, or replace with your custom plugin image
`vwnt.exe`	[Required] VisualWorks VM
`vwntoe.dll`	[Required] VisualWorks OE support
`vwplugin.inf`	Secondary INF installation file
`vwplugin.ini`	VisualWorks Plugin configuration file
`vwpluginax.inf`	Primary INF installation file
`vwpluginax.osd`	Main OSD installation file

To this set of files, add your own application parcels and/or replace

plugin-base.im with your custom image file. Make any necessary changes to the **vwplugin.ini** configuration file.

When all the necessary pieces of your application are in place, edit the **mkcabpluginax.bat** file to reflect the desired content of your CAB deployment file and edit **vwplugin.inf**, **vwpluginax.inf** and **vwpluginax.osd** to reflect the same content. It is critical that the version number is identical wherever it appears in the plugin DLL, the INF and OSD files. This is the version number which appears in the CODEBASE attribute of the OBJECT tag. The Microsoft MSDN Library article “How to Package Components for Internet Distribution” contains a good introduction to INF and OSD syntax and the parameters for the **CABARC.EXE** tool.

If you are building the deployment file for your own custom VisualWorks Plugin ActiveX Control (see the following section), replace the plugin DLL (**axvwplugin.dll)** with your own custom control and again ensure that all version specifications are identical.

Run the batch file (**mkcabpluginax.bat**) to create your own CAB deployment file. If you will be signing your control, insert the digital signature into your CAB deployment file.

You are now ready to test your plugin. Copy your CAB deployment file to your web server location and reference it (with the correct version) in the CODEBASE attribute of an OBJECT tag on your HTML page. If you are still testing with the generic plugin DLL, make sure you have removed the original installed version of the generic VisualWorks Plugin ActiveX Control (see above). When you open the HTML page, the web browser will download your new CAB deployment file and install your new plugin.

Building Your Own Plugin Control

Our generic VisualWorks Plugin ActiveX Control, as distributed, is not safe for initialization or safe for scripting. Since a VisualWorks application has complete access to the user’s machine, we have no way to guarantee that the generic plugin cannot be used for inappropriate purposes, however unlikely that might be. It is unfortunate but true in today’s world that ActiveX Controls, which are normally installed through Internet Component Download, are responsible for a lot of machine hacking and virus activity. There is a responsibility associated with marking an ActiveX Control as safe – you as the developer must ensure that your control really cannot damage a user's machine.

To deploy your own VisualWorks Plugin ActiveX Control, you will need access to the source code for our ActiveX Control. You will need to mark your control safe at the source level and rebuild the DLL. Since you will be changing Cincom's code to construct your own ActiveX Control, you must also take responsibility for the resulting DLL by

- Replacing the VisualWorks Plugin GUIDs with your own GUID identifiers
- Changing the DLL's version information to reference your company's name, copyright, etc.

You may not, however, remove the internal copyright information from the text of our C/C++ code.

The VisualWorks Plugin source code is covered under your VisualWorks license agreement. Currently you can request this source from VisualWorks Support through normal support channels. Once we have completed implementation of the new cross-browser NPAPI Plugin version which will run in newer versions of NPAPI enabled browsers such as Netscape, Mozilla, Opera and Firefox, the source will be included on the distribution CD. Until this work is finished, the plugin source code may change significantly from release to release.

The VisualWorks Plugin ActiveX Control is an ATL Control, built with the Microsoft VisualStudio .NET C++ compiler. You will need to purchase and install a copy of this compiler to build your custom control.

Implementing Object Safety

The VisualWorks Plugin DLL has little potential for damaging a user's machine on its own. It is simply the interface to connect the VisualWorks image and the web browser.

The DLL is responsible for allocating its own context, the contents of any POST request, plus a shared memory area also used by VisualWorks. It reads the initialization file and passes GET and POST requests to the web browser (Internet Explorer), caching the result as a local file when specified. These are not risk-free, and buffer-overflow is a common security hole with C/C++ code, but the primary risk potential is in your Smalltalk code.

The VisualWorks image maintains complete control over loading parcels as well as implementing the security measures defined by the ALLOW or DENY attributes of the initialization file. You must ensure that your applet,

as a combination of the ActiveX Control DLL plus the Smalltalk image and parcels, was tested in a variety of scenarios and abides by the following guidelines:

- do not manipulate the file system
- do not manipulate the registry (except to register and to unregister itself)
- do not overindex arrays or otherwise manipulate memory incorrectly
- validate (and corrects) all input, including initialization, method parameters, and property set functions
- do not misuse any data that is provided by the user or that is about that user

After you have completed sufficient testing to ensure that your control cannot be used to damage the user's machine, regardless of how it is intitialized or how it is scripted, the actual work to mark the plugin control safe for initialization and scripting is quite simple. This involves only a trivial change to the plugin source code. In short, you need to implement the IObjectSafety interface for the plugin control.

The plugin source contains a template for the IObjectSafety interface. See the source files **`VisualWorksAX.h`** and **`VisualWorksAX.cpp`**. Simply alter the template per the instructions in these files.

Updating DLL Version Information

The plugin source directory also contains **`VWPluginReadme.txt`** which details the remaining changes required to build a custom version of the control. These involve updating the version information to reflect your company's copyright and obtaining your GUID identifiers for the control. Follow these instructions, update the project properties to specify the name of your DLL, and rebuild the project.

Signing the Custom Plugin Control

The Microsoft MSDN Library article "How to Package Components for Internet Distribution" provides background information and instructions for adding a digital signature to your CAB deployment file.

Follow the instructions in the previous section to assemble the components of your application and build your CAB deployment file. Sign your file using the Microsoft Authenticode Tools. Once your application is packaged, make the CAB file available for download from your web server and reference it in the CODEBASE attribute of an OBJECT tag in your HTML page.

7

Communicating with a Plugin Application

Only the simplest applets run in a completely stand-alone manner. In general, an applet needs to communicate with other programs or with the browser itself.

The VisualWorks Plugin provides several options for allowing a plugIn application to communicate with other processes and applications.

- AppletModel provides protocol for doing HTTP GET and POST operations, and for updating the browser status line.
- Socket support is standard in VisualWorks, and can be used by the plugin.
- VisualWorks add-ins, such as the Database Connects, DST, and VisualWave are available for extending plugIn operations.

Get and Post operations are covered here, as well as the mechanism required to return focus to the main applet window after a child window closes. Other options are discussed in other VisualWorks documentation.

Doing Get/Post Operations

A common communication operation is to do a GET or POST operation in a browser, to send and receive data to a web server. For information on using these operations, refer to the W3 website, www.w3.org, or a publication on internet programming.

AppletModel provides protocol for making these calls from your applet. The full set of methods is provided in the pluginplugin api method category in the AppletModel class. To have access to these methods, your applet

needs to be a subclass of AppletModel. The methods are distinguished by which operation is to be performed, and the format in which data is presented and received back.

Using GET Messages

HTTP GET messages retrieve whatever data is identified by a URL. GET messages are also used for many search requests. The data is often an HTML page, a graphic, or a file.

The VisualWorks plugIn provides several messages to make it simple to issue a GET command from an applet.

For example, the getURL:target: message retrieves the contents of a URL and displays it in a targeted browser or frame. So,

```
self getURL: url target: '_blank'
```

contacts the site named in the url variable, and displays it in a new browser window (the _blank keyword is a reserved name in the HTML spec for this purpose). The target name may also be any named browser window or frame.

To process the content returned by the URL within the applet, you can use the message:

```
self getURLAsString: url
```

The contents of the URL, typically HTML source, can then be processed within the applet. For example, you may want to process the input using the XML support provided in VisualWorks. If a special encoding is needed, use getURLAsString:encoding:.

Note that if a GET causes the page containing the applet to be replaced or lost, as would:

```
self getURL: url target: '_self'
```

then the plugIn application is closed. Using the browser's BACK command does not restore the applet to its prior state, since the applet will need to be relaunched. When all plugIn applications are closed, the image itself exits.

Using POST Messages

An HTTP POST message is used for a variety of operations that submit data to an internet service, such as for:

- Annotating existing documents

- Posting a message to a bulletin board topic, newsgroup, mailing list, or similar group of articles
- Adding a file to a directory
- Extending a document during authorship

The VisualWorks Plugin provides several messages to make it simple to issue a POST command from an applet, distinguished by the type of data the applet is sending and how the return data is to be handled.

For example, this message transmits data as it would be sent from an HTML form:

```
self postToURL: url form: aDictionary
```

The Dictionary data is presented as key/value pairs, as expected from a form submission. The returned data is a String.

To post data from a string, use a form of the postToURL:string: message.

Refer to the plugin api message category for a complete set of POST messages. The message comments also mention special requirements and differences between MS Internet Explorer and Netscape Navigator.

Posting a String or a Form

When posting a string or a form, the headers are built automatically, if needed.

Posting Bytes

When posting bytes, you must provided headers as needed and use LF terminated lines. The ability to post bytes is for applications that need total control.

Returning Focus to the Main Applet Window

The window in the web browser which contains your running VisualWorks applet is really two windows working together. The first (parent) window is the window managed by the VisualWorks Plugin ActiveX Control (the DLL). The browser acts as a container for this window. The second (child) window is the window used by VisualWorks in which to display its GUI. The browser does not manage this window or know anything about its state.

If your application opens VisualWorks child windows, you need to explicitly re-activate the plugin control window if you want your main applet window to get focus after a child window closes.

To ensure that you return focus to the plugin applet correctly you must do two things:

- Your child window must be implemented using a model which is a subclass of ApplicationModel, so you can catch the close notification with an override of noticeOfWindowClose: which contains code to reactivate the plugin applet.
- Your subclass of ApplicationModel should store its parent (the instance of the plugin applet) which must be either a subclass of AppletModel or a subclass of ApplicationModel which implements the necessary plugin API (from **`plugin\applet-api.st`**). See “Adapting an Existing Application” in Chapter 3, “Developing a Plugin Applet.”

The Loan Calculator example in **`plugin\examples\pcl\LoanCalculator.pcl`** and executed with **`plugin\examples\loancalc-x.html`** demonstrates this functionality.

8

Plugin Tips and Tricks

There is very little that you can do in VisualWorks that you cannot do with the plugin as well. In this chapter we will highlight, by way of example, a variety of effects you can achieve with the plugIn.

Launch a Stand-alone Application

Because you have the full functionality of VisualWorks in the plugin image, you can launch a separate VisualWorks application from an applet. The launched application can then run in complete independence of the browser.

Since the stand-alone application has no dependencies on the browser, you can create it as a subclass of ApplicationModel. If you have several such applications already, they can be presented by the applet as a list, with each list item launching the selected application.

Dynamic Component Update

To update HTML pages, including such items as menu selection lists or other display items, requires downloading and redisplaying the entire page. For items that can be based on information at the client, using a VisualWorks applet is much faster.

For example, an applet could include a Notebook widget and all the related logic. Changing a notebook page can change the whole look of the applet. Since the entire change is in VisualWorks, it is much quicker than loading and drawing a new HTML page.

Lists can also be dynamically updated using data selected in the applet itself, or items entered in an entry field.

Menu selections can also be defined entirely within the VisualWorks applet, and so expanded much more quickly than is possible using typical web-page techniques.

In general, the more you can accomplish within the applet itself, the more you can do on the client system and accomplish with the speed of VisualWorks.

Access Files

VisualWorks applets do not impose any of the artificial restrictions familiar to those who have attempted writing applets under Java. For example, file read/write access is open in the plugin environment using the standard VisualWorks file access API.

This facility is very powerful, and is potentially destructive because files can be overwritten, posing certain security risks. Please use this capability carefully and responsibly.

Index

Reader Comment Sheet

Name: _____

Job title/function: _____

Company name: _____

Address: _____

Telephone number: ( ) - Date: / /

How often do you use this product? ❑ Daily ❑ Weekly ❑ Monthly ❑ Less

How long have you been using this product? ❑ Months ❑ Years

Can you find the information you need? ❑ Yes ❑ No

Please comment. _____

_____

Is the information easy to understand? ❑ Yes ❑ No

Please comment. _____

_____

Is the information adequate to perform your task? ❑ Yes ❑ No

Please comment. _____

_____

General comment: _____

_____

_____

_____

To respond, please fax to Larry Fasse at (513) 612-2000.

CINCOM®
The Smart Choice®