Cincom

VisualWorks®

Web GUI Developer's Guide

P46-0139-00

Part Number: P46-0139-00

Software Release 7

This document is subject to change without notice.

RESTRICTED RIGHTS LEGEND:

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Trademark acknowledgments:

CINCOM, CINCOM SYSTEMS, and the Cincom logo are registered trademarks of Cincom Systems, Inc. ParcPlace and VisualWorks are trademarks of Cincom Systems, Inc., its subsidiaries, or successors and are registered in the United States and other countries. ObjectLens, ObjectSupport, ParcPlace Smalltalk, Database Connect, DLL & C Connect, COM Connect, and StORE are trademarks of Cincom Systems, Inc., its subsidiaries, or successors. ENVY is a registered trademark of Object Technology International, Inc. All other products or services mentioned herein are trademarks of their respective companies. Specifications subject to change without notice.

The following copyright notices apply to software that accompanies this documentation:

VisualWorks is furnished under a license and may not be used, copied, disclosed, and/or distributed except in accordance with the terms of said license. No class names, hierarchies, or protocols may be copied for implementation in other systems.

This manual set and online system documentation © 1993–2002 by Cincom Systems, Inc. All rights reserved. No part of it may be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior written consent from Cincom.

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 guide is designed to help VisualWorks programmers create web-based applications effectively using VisualWave. VisualWave is a server technology supported by the VisualWorks Application Server.

This document accompanies the *VisualWorks Application Developer's Guide*, and the *VisualWorks GUI Developer's Guide*, which together provide information that will help you effectively use the features in VisualWorks Application Server.

Audience

The discussion in this book presupposes that you have at least a moderate familiarity with object-oriented concepts and the VisualWorks environment. It also presupposes that you have a good understanding of the World Wide Web, web (HTTP) servers, browsers, and HTML.

In addition to this book, the documentation set for the VisualWorks Application Server includes the following:

- *Web Application Developer's Guide:* Provides detailed information about building Web applications using the VisualWorks Web Toolkit.
- *Web Server Configuration Guide:* Provides more detailed information about installing and configuring server applications, the internal architecture of the VisualWorks Application Server, and its interface with commercial HTTP servers.

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.
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 commericial 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

VisualWorks includes an online help system.

To display the online documentation browser, open the **Help** pull-down menu from the VisualWorks main menu bar and select one of the help options.

Commercial Publications

Smalltalk in general, and VisualWorks in particular, is supported by a large library of documents published by major publishing houses. Check your favorite technical bookstore or online book seller.

Examples

VisualWorks Application Server includes many examples, including some which are not referred to in this manual.

Examples demonstrating the use of server pages and servlets can be found in the **`\web\examples\`** directory.

To explore the UI Painter examples, load the demonstration parcel, **`VisualWaveDemos.pcl`**. Specific examples are referred to by class name.

1

Overview

The VisualWorks Application Server is a full-featured environment for creating and maintaining Web business applications using VisualWorks. A flexibile, scalable architecture provides support for industry standard Web technologies, and support for three distinct application frameworks: Smalltalk Server Pages, Servlets, and VisualWave.

For a general overview of the VisualWorks Application Server, or for details on building applications using Smalltalk Server Pages or servlets, refer to the *VisualWorks Web Application Developer's Guide*.

This guide covers the details of building applications using VisualWave. Existing VisualWorks applications may be easily Web-enabled using VisualWave, though typically some modifications are required to optimize an application for the Web.

By nature, Web applications impose certain restrictions on the application developer. This document addesses these restrictions, as well as special enhancements that allow you to taylor existing VisualWorks applications for the Web.

This chapter describes:

- Building Web Applications
- Design Considerations
- Supporting Graphics
- Supporting Database Access

Building Web Applications

VisualWave enables developers to build Web applications using the standard VisualWorks painting paradigm:

- Design the application domain model using an object-oriented architecture.
- Create or adapt classes that define the data and behavior of the application objects (the application model).
- Build the user interface using the UI Painter and related editor tools.

With VisualWave, the difference occurs at runtime. Where VisualWorks uses a window specification to create an onscreen display, VisualWave uses it to define an HTML (Hypertext Markup Language) page.

When building window specifications, VisualWave adapts the VisualWorks UI Painter in several ways:

- Providing Special Widgets
- Excluding Some Unsupported Widgets
- Excluding Some Unsupported Widget Properties
- Providing Look Policies

The following chapters in this book discuss the additions provided by VisualWave as well as the restrictions and other differences imposed when building Web applications.

Design Considerations

When designing and building a VisualWorks application that will run on the Web, the following points should be kept in mind:

Web applications are multi-user

Whereas client-based applications are typically single-user, multiple users will be running instances of your Web application simultaneously. Applications built for VisualWave should be thread-safe.

Navigation within the application must be open

Web browsers make it possible (using the back and forward buttons) to violate the application's assumptions of where a user can go in the application at any given time. It is always the client that initiates communication and s/he may switch to pages in an open pattern.

Web applications use forms

In a VisualWorks application, users enter information in one widget at a time. In a Web application, users enter information in several widgets and then submit all the new values in batch mode. Applications running with VisualWave process those new values according to tab sequence order.

Dynamic widgets may present difficulties

Because of the "fill in and submit" nature of HTML forms, Web applications should avoid some of the dynamic capabilities of screen-oriented GUIs such as immediate modification of widget properties based on user input. Features such as hierarchical menus do not work well.

Some widgets that are common in screen-oriented applications are not readily represented by Web browsers.

Screen layout is only approximate

Web browsers are not WYSIWYG. In fact they're intentionally just the opposite, allowing you to separate content from presentation.

When using the UI Painter to create a Web application, it is not possible to achieve the kind of precise layout to which most GUI designers have become accustomed. Having said that, there are tricks that can help. For example, you can group widgets together in the painter in order to line them up better. Experimentation and preview are key.

For an application that requires sophisticated layout, you should consider using commerical Web design software in conjunction with Smalltalk Server pages.

Supporting Graphics

Web applications built using VisualWave require a server that supports graphics. Accordingly, a VisualWave application cannot be “headless” (lacking a GUI), as can a VisualWorks or Web Toolkit application.

VisualWave uses resources on the host machine when converting images to GIF or GIF-compatible (GUF) format for display on the client’s web browser. As a result, if the server machine has a monochrome display, then only monochrome images are supported to the client.

> **Note:** Because GIF and GUF graphics are 8-bit, we strongly recommend running the host machine’s window system in 8-bit mode.

Class Image does not support transparency in graphics. To create transparent images, save the image object as an external GIF file and use a standard graphics editor to make it transparent.

Supporting Database Access

To deploy a database application as a web application, you can use the ObjectLens programmatically as described in the *VisualWorks Database Connect Application Developer’s Guide*. You can also use the Data Modeler. You cannot, however, use the data forms or canvases that are generated by the database tools. The dataset, linked data form, and embedded data form widgets are also not supported.

Database applications that use the EXDI and Database Connect technology directly are supported for both web and non-web applications. Note, however, that multiple clients using the application can cause performance and concurrency issues on platforms that do not support the Threaded API. For details, see the *VisualWorks DLL & C Connect User’s Guide*.

2

Windows and Applications

Using multiple windows is a common technique in VisualWorks GUI design. The most common additional windows are dialogs. Others may be opened for providing alternative views on application data.

Multiple windows need to be handled differently by a VisualWave application than in VisualWorks.

An alternative to multiple windows in some cases is to use frames. For more information on using frames in a VisualWave application, refer to "HTML Frames" on page 47.

This chapter describes:

Windows and Sessions

Normally, a VisualWorks application opens another window by sending open to that window's application model:

```
actionButton
    ApplicationModelName open
```

When a VisualWave application starts, it automatically creates a *web session* for itself. The web session keeps track of the client host and web browser to which the application should send its output. All of the application's windows must belong to the same session. VisualWave extends the VisualWorks application model by adding protocol that enables the first application model to open other application models and assign them to its web session.

Any of the following will work to replace the example above:

```
actionButton
    self openClass: ApplicationModelName

actionButton
    self openClass: ApplicationModelName interface: #mySpec

actionButton
    | am |
    am := ApplicationModelName new.
    am setSomething: nil.
    self open: am interface: #windowSpec
```

When converting a screen-oriented application for use with VisualWave, it is important to consider all continuity across the user interface. If the web-based application requires session continuity from page to page, it is tempting to use a nested subcanvas in the screen-oriented application. However, because of the implementation of web sessions, care must be exercised so that the nested subcanvas object doesn't lose its context.

VisualWave provides a mechanism for keeping track of the *phase* of a web session. If the application presents a series of pages, the VisualWave application server notes which page the client is on, and requires that **`FORM`** data be received only from that page.

This mechanism ensures the continuity of a web session. If, for example, the client uses the Web Browser's **Back** button to look at a previous page, and then tries to **`SUBMIT`** input from that page, VisualWave will detect this and signal an *out-of-phase condition*. By default, the server reports an error, but it is possible to override this action, and selectively give the responsibility for maintaining session phase to the application.

Web Sessions and Phasing

VisualWave maintains the state of a client's interaction with an application using the notion of a web session. The application typically presents a series of pages to the client, each of which may be submitted as **FORM** data. If the client uses the Web Browser's **Back** button to move to a previous page in the session, and if the previous page is then submitted with stale data, VisualWave will raise an out-of-phase condition, and report an error to the client.

You can tell VisualWave to ignore this out-of-phase condition, and allow your application to make a decision about the best response to a phase error. For maximal control, you can enable and disable trapping for the out-of-phase condition on a widget by widget basis.

To disable reporting of an out-of-phase condition, select the widget you wish to change, open a Properties Tool, and set the **Ignore Phase Errors** check box on the **Web** properties page.

Using Dialogs

VisualWave supports the use of some Smalltalk dialogs. You can also implement web dialogs using JavaScript.

VisualWave displays dialogs in the client's web browser like any other web page. The user must, however, respond to the dialog before entering information on any of the application's pages. Attempts to use another page return the user to the page containing the dialog.

The following expressions can be used to open a dialog:

```
self openDialogInterface: #dialogSpec

self dialog warn: 'Do not do that'

Dialog warn: 'Do not do that' for: self builder window
```

To set the look policy, session, and builder:

```
self simpleDialog
```

To have the application set the look policy and session for the dialog:

```
SimpleDialog new
```

User Interface Design Considerations

Keyboard hooks, auto accept, and direct manipulation of controller and view state occasionally performed in VisualWorks applications are not available to web applications. Similarly, in a VisualWave application, `componentAt:` returns the view itself, not the wrapper.

Visual blocks for sequence views, such as graphics in tables and lists, are not supported.

Care must be exercised when using nested subcanvases in several parts of an application. Under some conditions, the application can break when a subcanvas is invoked.

3

Widgets

VisualWorks provides a rich set of widgets for building applications and VisualWave adds a number of special widgets to the standard palette. Due to limitations of HTML and HTTP, a few standard widgets are not supported in VisualWave.

This chapter provides an overview of working with widgets in VisualWave. Later chapters include additional details on specific widgets.

VisualWave Widgets

HTML Text Widget

A text field that allows you to specify font changes, anchors, and links. For the Aspect property, you can supply text, a URL, or a message that returns HTML text. See “Creating HTML Text” on page 37 for details.

VRML Widget

An enhanced view holder that allows you to incorporate a VRML scene in a VisualWave application. See “VRML Widget” on page 83 for details.

Java Applet

An enhanced view holder that allows you to incorporate a Java applet in a VisualWave application. See “Java Applet” on page 89 for details.

ActiveX Widget:

An enhanced view holder that allows you to incorporate an ActiveX control in a VisualWave application. See “ActiveX Widget” on page 135 for details.

Supported Widgets

Not all standard VisualWorks widgets have HTML equivalents. Widgets that have no HTML equivalent become invisible and inactive in Web applications.

The following table shows which widgets are supported in Web applications and, of those, which can have alignment properties set.

Widget	OK for Web	Can Align
Action button	X	X
ActiveX control	X	X
Chart	X	X
Check box	X	X
Click map	X	X
Combo box		
Dataset		
Divider (horizontal)	X	X
Divider (vertical)		N/A
Group box		
HTML text	X	X
Input field	X	X
Java applet	X	X
Label	X	X
List view	X	X
Menu button	X	X
Notebook		
Percent Done		
Radio button	X	X
Region		
Resizer		
Slider		
Subcanvas	X	X
Tab Control		

Widget	OK for Web	Can Align
Table	X	X
Text editor	X	X
Tree View		
View holder	X	X
VRML	X	X

The following widgets generate errors in Web applications:

- Embedded data forms
- Linked data forms

Unsupported Widget Properties

Some widget properties are ignored because they cannot be specified precisely for all web browsers.

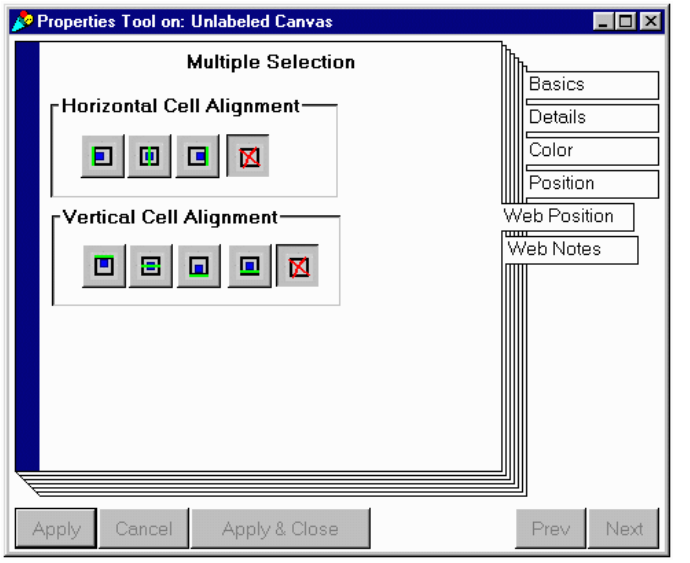
See the **Web Notes** in the widget properties tool for details about specific widgets and their properties. In general:

- **Position** properties are ignored, except those set using the **Web Position** page of the properties tool.
- **Validation** properties, except **Change**, are ignored.
- **Notification** properties, except **Change**, are ignored.
- **Read-only** widgets appear to be editable. They are read-only in implementation; the controller does not process anything the user enters into them.
- **Enable/Disable** has the same effect as beVisible/beInvisible.
- Table cells that are selectable are displayed with radio buttons. For example, if you have the cell-select property chosen for a table, every cell will have a radio button in it. Currently, VisualWave does not support non-select tables. Cell, row, or column selection must be chosen.

Controlling Widget Layout

When VisualWave generates a web page under the **Enhanced HTML** look policy, it uses a table to control the layout of all of the widgets on the canvas (for details on look policies, "Look Policies" on page 29). Each widget is placed in its own table cell in the HTML page.

The alignment of each widget within its table cell is controlled using the **Web Position** properties:



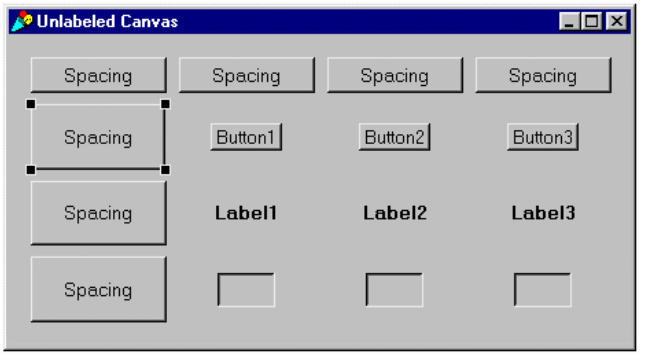
You can control both the horizontal and vertical alignment. Use the radio buttons to set the horizontal alignment to left side, center, right side, or none. Similarly, a widget's vertical alignment can be set to top, middle, bottom, baseline of text, or none.

For a single widget on a page, the alignment makes no difference because that widget determines the cell size. When multiple widgets cooperate to determine the width of a column or the height of a row, the alignment can make a significant difference.

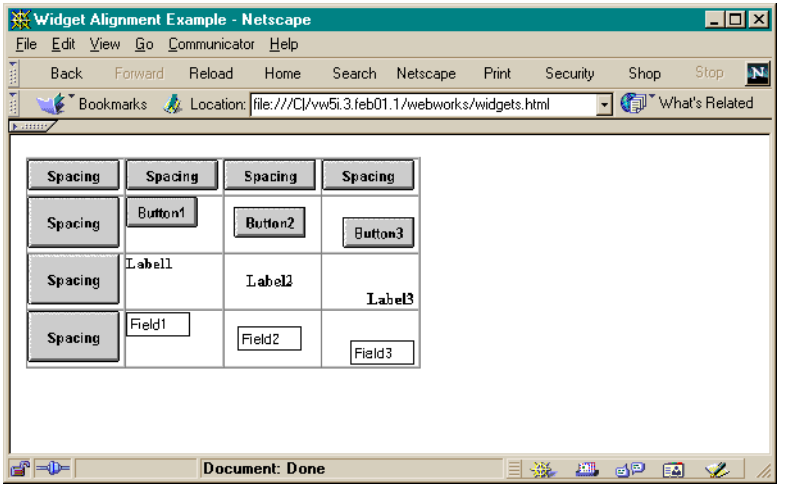
To see how alignment works, set up a canvas with a series of tall and wide "spacing" widgets that determine the size of the rows and columns. Then add widgets that are smaller than the spacing widgets and adjust the alignment of those smaller widgets.

In the canvas below, **Spacing** buttons are used to determine the column and row sizes. The other buttons, labels, and input fields appear to have the same alignment on the canvas, but their properties are set so that:

- **Button1**, **Label1**, and **InputField1** are aligned to the left and top.
- **Button2**, **Label2**, and **InputField2** are aligned to the center and middle.
- **Button3**, **Label3**, and **InputField3** are aligned to the right and bottom.



The resulting HTML page is shown here:



Notice that table borders are turned on to show the alignment. You can turn table borders on for VisualWave by sending the message:

```
NetscapeTableLayout showBorders: true
```

For a list of widgets that can be aligned, see page 24.

Layout Tips

The painting tools are used to specify the basic layout that will appear in the web browser. They do not, however, describe the exact layout.

One of the benefits of HTML is that it separates content from presentation. You specify the kinds of elements that you want on your web page and each web browser then determines how to present those elements. Therefore, the layout you see on the web browser will only be an approximation of what you drew in the Painter. However, there are ways to control the display.

Window Size

If you change the window size, you must set the new window size as the preferred size using **Layout → Window → Preferred Size**.

Widget Size

The height of the following widgets is not affected by the position properties when they are displayed in a web browser:

- Action button
- Check box
- Radio button
- Input field
- Menu button

The width of action buttons, check boxes, and radio buttons is determined by the label of the widget. Setting the position properties does not affect display in a web browser.

Widget Placement

Use the Painting Tool's **Group** command (on the **Arrange** menu) to line up widgets better.

Canvas Properties

The **Web** properties page for a canvas allows you to set a background pattern and sound, as well as a few other properties that affect the behavior or appearance of the web page displaying your application. Refer to the **Web Notes** page in the Properties tool for more information.

Look Policies

The painting tools support two web-specific look policies:

- **Basic HTML** is used for web browsers that do not support tables.
- **Enhanced HTML** is used for web browsers that do support tables.

To set the look policy, open the Canvas Tool's **Look** menu. When you choose an HTML look policy, the canvas redraws itself to look more like it will when displayed in a web browser. Widgets for which there are no HTML equivalents become invisible and inactive.

When the application is running, VisualWave automatically determines which look policy to use. With a VisualWorks Application Server image, you can specify the exact set of browsers for which each HTML look policy is used.

4

Creating an Image Map

The Click Map widget is a key element in the VisualWave implementation of HTML image maps. A click map allow you to define areas of a graphic image and determine what action to invoke based on the area in which the user clicked.

Click maps are often used for button bars at the top or bottom of web pages. Using a click map makes it possible to use any graphic image to represent a row of buttons. The image may include text, graphics, and a variety of colors. All of the buttons may be part of a single image, guaranteeing their relative layout and allowing for a wider variety of layouts than permitted by HTML button entities. Quite often button bars created with click maps are highly stylized with elaborate graphic images and text.

Setting Up a ClickMap

To create a click map:

1. Load or create the background graphic image
2. Add a click map widget to the canvas
3. Program the click map widget
4. Define the hot regions for the click widget

You can also use a custom view (see: “Using Custom Views and Controllers” on page 36).

Loading or Drawing the Background Image

The VisualWorks Image Editor enables you to draw graphic images and store them in resource methods. The Image Editor is available from the **Tools** menu of the VisualWorks Launcher window and the **Tools** menu of the UI Painter. It is described in the *VisualWorks Application Developer's Guide*.

VisualWave also allows you to load bitmap graphics that have been drawn with standard drawing and painting programs. To load a bitmap into your image:

1. Open a Resource Finder by choosing **Browse→Resources** in the Launcher window.
2. In the class list (left-hand list) of the Resource Finder, select the class in which you wish to store the image. This must be an application model class.
3. In the resource list (right-hand list), select **new Image From File** on the <Operate> menu, enter the name of the bitmap file, and click **OK**.
4. Enter the selector you want used for the method that stores and returns the bitmap image resource. Click **OK**.

VisualWave reads the bitmap image from the file provided and converts it into a Smalltalk graphic image. The specification for the Smalltalk graphic image is stored in a newly created resource method.

Adding a Click Map Widget to the Canvas

The graphic image is displayed on the canvas by a click map widget. The click map widget also contains the hot region mappings to use with the image. This widget is the connection between the image and the actions that user clicks will invoke.

To add a click map widget to a canvas:

1. Open a canvas for editing.
2. From the Palette, choose the click map widget: 
3. Place the click map on the canvas.

Programming the Click Map Widget

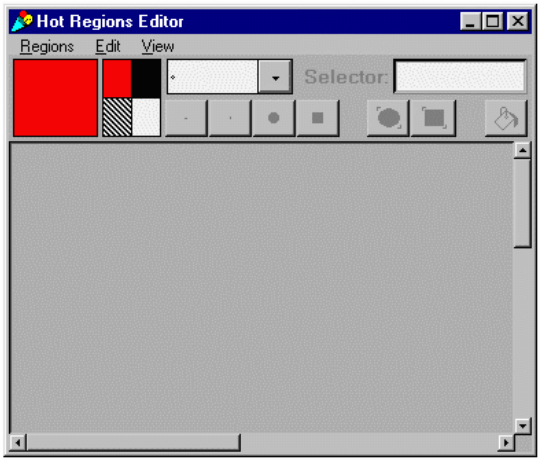
The click map widget needs to know which image to display as the background for the hot regions. You specify the click map's image and other information through the Properties Tool:

1. With the click map selected, click the **Properties** button in the Canvas Tool or choose **properties** from the canvas's <Operate> menu.
2. In the Properties Tool, set the following **Basic** properties:
   - **ID**: A unique identifier for the widget.
   - **Visual Message**: Determines the image that is displayed. It can be the selector for the resource method in which the image is stored. It can also be an URL that directs your web server to an external image file.
   - **Default Click Message**: (optional) Determines the action to be invoked when the user clicks in an area for which no other action is specified.
3. Apply the properties to the widget by clicking the **Apply** or **Apply and Close** button. The image appears in the click map widget.
4. In the canvas, resize the click map widget so that it closely outlines the graphic image.
5. If you changed the size of the canvas, set the new window size by choosing **Layout→Window→Preferred Size**.
6. Install the canvas by clicking the **Install** button or choosing **install** from the canvas' <Operate> menu. When prompted, provide the class name and method name in which to install the canvas.

Defining the Hot Region Mappings

1 Make sure the click map widget is selected in the canvas.

2 In the Canvas Tool, choose **Tools→Hot Regions Editor**.



3 To display the selected click widget's image as the background in the Hot Regions Editor, choose **Regions→Read**. Resize the editor window to display the entire image.

Alternatively, you can choose **View→Load Backdrop** and specify the class and selector from which to read the image.

4 To start defining a region, choose **Edit→New Slice**.

A slice is comprised of one or more areas that invoke the same action when clicked. The areas in a slice do not need to be contiguous. For example, you could have the area at the left end and the area at the right end be the same slice and invoke the same method.

You specify the areas that make up a slice by painting. The Hot Regions Editor contains basic painting tools similar to other bitmap editors. There are:

- Four colors or patterns of ink, allowing you to choose the color that gives you the most accuracy when painting regions on your image. The color does not show up on the final click map.
- Six brush sizes and shapes. The first four draw lines, the next draws an ellipse, and the sixth draws a rectangle.
- A fill-mode bucket that fills the entire image.

5 Paint the slice.

6 In the **Selector** input field, enter the selector for the action method that will be invoked when this slice is clicked.

7 Press <Return>. The selector appears on the menu button in the middle of the Hot Regions Editor. You can use the menu to switch between different slices of the same hot region resource.

8 Repeat steps 4 through 7 to define all the slices for this hot region.

9 Choose **Regions→Apply** to:

- Install the hot region resource. You are prompted for the class name and selector for the resource method that stores the hot region mappings.
- Insert the resource's selector into the **Mapping Selector** property of the click map widget that is selected in the canvas.

10 Close the Hot Regions Editor.

11 Install the canvas.

Click Map Widget Properties

Visual Message
This property determines the graphic image that is to be displayed as the background for the click map.

Mappings Selector
This property specifies the method that performs the actual mapping between the area in which the user clicked and the appropriate action for that location. You define the areas and associate them with methods to invoke by using the Hot Regions Editor.

Default Click Message
This property specifies the method to invoke when the user clicks in an area for which no other method has been specified.

Note that it is the click map widget that makes the association between a graphic image and a set of hot regions. The graphic image and the hot region mappings are not stored together. They are associated by virtue of being named in the click map's **Visual Message** and **Mappings Selector** properties. Thus, you could use reuse the graphic image with different hot region mappings.

Hot Regions Editor

The Hot Regions Editor is used to create and edit a hot region mapping, which can be integrated into a click map widget. You use the Hot Regions Editor to create hot regions and specify messages to be sent when the user clicks in those regions. The Hot Regions Editor uses your entries to generate a specification for building an appropriate hot region object. This code is then installed in an application model as a resource method.

To open the Hot Regions Editor, open the Canvas Tool and choose **Tools→Hot Regions Editor**.

For a complete functional description of the Hot Regions Editor, see "Hot Regions Editor" on page 139 in Appendix A.

Using Custom Views and Controllers

If you have an application that uses custom views and controllers, you can make it work as a VisualWave application:

1. Make the custom view a subclass of ClickWidget.
2. In the custom view class, implement a mouseReleaseAt: method that specifies what to do when the user clicks in the custom view.
3. Disconnect the custom controller from your application. Move any custom behavior to the custom view's mouseReleaseAt: method.

Remember that interaction with the user is more limited in web applications than in other applications. In particular, the only mouse events transmitted to the application from the web browser are mouse clicks. Events based on entry, mouse movement, and exit are ignored.

5

Creating HTML Text

This section describes how to add an HTML text widget to the canvas, and how to create and display the HTML text. It explains how to create HTML text both programmatically and using the HTML Text Editor.

HTML Text Widget

The HTML text widget allows you to place arbitrary pieces of HTML in your application. It is essentially a text field that allows you to specify font changes, anchors, and hyperlinks.

The **Aspect** property determines where the widget looks for the item to display. For the **Aspect** property, you can supply text, an URL, or the name of a method in the application model class that returns an instance of HTMLText. The method may be an instance method that you have programmed to return HTML text or a resource method that you created with the HTML Text Editor.

The HTML text widget's properties are similar to those of a regular text field (for details, see the *VisualWorks Application Developer's Guide*). Note that some properties are not supported in web browsers. See the **Web Help** properties page for details.

Adding an HTML Text Widget

To display HTML text, place an HTML text widget on the canvas and specify its properties. The properties include the name of a method that provides the HTML text to display.

To add an HTML text widget:

1. Open a canvas for editing.
2. In the Palette, select the HTML Text widget: 
3. Position the mouse pointer on the canvas and click to place the HTML text widget.
4. Size and shape the widget as desired. Keep in mind that you are only providing general instructions for the web browser. The web browser will display the widget as tall and wide as is necessary to display the widget's entire contents.
5. With the HTML text widget still selected, click the **Properties** button in the Canvas Tool or choose **properties** from the canvas's <Operate> menu.
6. In the Properties Tool, set the **Basic** properties:
   - **Aspect**: The name of the method that returns an instance of HTMLText. This may be the name of an instance method that returns an HTMLText object programmatically or the name of a class method that returns an HTML text resource that was created with the HTML Text Editor.
   - **ID**: A unique symbolic name for the HTML text widget.
7. Apply the properties to the widget by clicking the **Apply** or **Apply and Close** button.
8. Install the canvas by clicking the **Install** button or choosing **install** from the canvas's <Operate> menu. When prompted, provide the class name and method name in which to install the canvas.

Providing HTML Text for the Widget

The widget uses the **Aspect** property to obtain its HTML text:

- If the application model for the canvas has an instance method whose selector matches the **Aspect** property, that method is executed. The result is displayed in the HTML text widget. The instance method may return an instance variable that holds onto the HTML text.

- If an instance method is not found, a check is made for a resource method with that selector on the class side of the application model. If a resource method exists, its resource is displayed in the HTML text widget.

To use an instance method, you programmatically create the method and the HTML text. To use a resource method, you use the HTML Text Editor to create the method and HTML text. Using an instance method to programmatically generate and return HTMLText is acceptable for very small pieces of text. For larger pieces, you may want to create a resource.

If you use an instance method, you can easily create a value holder on the text and store the value holder as an instance variable. You can send value: to the instance variable with a new piece of HTML text, and it will change in the widget. This strategy is demonstrated in *Getting Started with VisualWave* for message text that changes depending on the context.

Programmatically Creating HTML Text

To use an instance method to supply HTML text, create an instance method whose selector matches the widget's **Aspect** property and then program that method to return an instance of HTMLText.

Generating a Stub using the Painting Tools

If you are going to use the instance method in combination with an instance variable, you can use the painting tools to get started:

1. In the canvas, select the HTML text widget.
2. Click the **Define** button in the Canvas Tool.
3. The Definer asks you to verify that you want to define a model (combination instance variable and instance method that returns that variable's value model) and add initialization.
4. Click **OK** to confirm the Define dialog.

   VisualWave adds the following to the application model class:

   - An instance variable (whose name matches the **Aspect** property) that holds onto the HTML text that the widget displays.
   - An aspect method (whose selector matches the **Aspect** property) that returns the value of the notifier instance variable.

Creating the HTML Text Instance

Whether you start from the painting tools or start from scratch, you need to edit the instance method to return an instance of HTML text for the words and formatting you desire. You can create HTMLText using one of three methods:

```
String>>asHTMLText
HTMLText>>fromString:
HTMLText>>convertFromString:
```

Sending asHTMLText to a string is a quick way to turn the string into an instance of HTMLText.

If your string already contains HTML markup tags, use the asHTMLText or fromString: method.

If your string does not contain HTML tags, use the convertFromString: method. When HTMLText converts a string, it encodes all HTML special characters (such as <>&"). To add additional HTML markup, you send messages to the HTMLText. HTMLText inherits protocol from its superclass Text that enable it to add emphasis to individual characters.

For example, suppose that you wanted to have HTML text that displayed the location of Cincom's web site, www.cincom.com. You want all of the text to be bold and be a link to the web site, and you want the "www" to appear in italics. The following code creates the appropriate instance of class HTMLText and assigns it to the temporary variable myHTMLText:

```
| myHTMLText |
myHTMLText := 'www.cincom.com' asHTMLText allBold.
myHTMLText emphasizeAllWith:
      (#anchorDestination-> 'http://www.cincom.com').
myHTMLText emphasizeFrom: 1 to: 3 with: #italics.
```

When the text is asked to "display" on a web browser, the emphases are converted to HTML tags. The HTMLText defined above would result in this HTML markup:

```
<A HREF=http://www.cincom.com>
<B><I>www</I>.cincom.com</B></A>
```

And the text will be rendered as a bold anchor with "www" in italics.

Supported HTML Tags

To view the currently supported HTML tags, open a System Browser on the class HTMLCharacterTag. Look on the class side in the protocol **class initialization** for the method initializeEncodings. This method contains expressions of the form:

```
Encodings at: #H1 put: <H1> -> <H1>.
```

The code above translates the emphasis #H1 to the HTML **`<H1>`** tag. For example, if you want to display Cincom's web location as a heading, you evaluate this code:

```
| myHTMLText |
myHTMLText := 'www.cincom.com' asHTMLText.
myHTMLText emphasizeAllWith:
      (#anchorDestination- 'http://www.cincom.com').
myHTMLText emphasizeallWith: #H1.
```

HTML Text Editor

The HTML Text Editor is used to create and edit HTML text *resources*, which can be integrated into an HTML text widget. You use the HTML Text Editor to create text and specify characteristics for that text. The HTML Text Editor uses these entries to generate a specification for building an appropriate HTML text object. This specification is then installed in a resource method on the class side of the application model.

To open the HTML Text Editor from a Canvas Tool, choose **Tools→HTML Text Editor**.

For a complete functional description of the HTML Text Editor, see "HTML Text Editor" on page 142 in Appendix A.

Using the HTML Text Editor

The text editor is used to create an HTML text resource.

To use a resource method to supply HTML text, you create a resource with the HTML Text Editor and install it on a method whose selector matches the widget's **Aspect** property.

Opening the HTML Text Editor

You can open the HTML Text Editor from either the Launcher window or the Canvas Tool by choosing **HTML Text Editor** from the **Tools** menu.

If you open the HTML Text Editor from a canvas that has already been installed, the HTML Text Editor uses information from that canvas to set defaults for certain operations, such as installing. If an HTML text widget is selected, you can read and apply properties between the widget and the HTML text resource.

Editing and Formatting HTML Text

In the HTML Text Editor, you can now enter some text, highlight it, and apply various emphases to the highlighted text, including anchors. To remove emphases, select the text, and deselect the emphases you want removed. Use the **X** button to remove all emphases.

Click on the **show HTML** button to view the HTML. While in this mode, you can enter arbitrary HTML markup. You cannot, though, create ordered lists directly with the editor. While in the **show HTML** mode, however, you can type in an ordered list by hand. If you save the HTML text while in **show HTML** mode, your markup will be preserved. If you select **show text**, you will lose all direct HTML editing changes.

Connecting your HTML Text to the Widget

When you are finished, choose **HTML→Apply** in the HTML Text Editor.

If an HTML text widget is selected in the canvas, this action applies the text in the editor to that HTML Text widget. To make the text appear, reapply the **Aspect** property in the Properties Tool. The text should now appear on the canvas.

In the **show HTML** editing mode, the text appearing in the canvas has HTML tags in it, whereas in the **show text** mode, rendered text appears on the canvas.

Importing HTML Markup

You can import HTML source code from either a local file or from a URL.

To import HTML markup from an external file on your system. choose **File→Open…** in the HTML Text Editor and enter a file name.

To import HTML markup from the web:

1. In the HTML Text Editor, choose **File→Open from URL…**.
2. When prompted, enter the URL for the desired file and click the **OK** button. The URL must be an HTTP request; it cannot be an ftp, gopher, or other kind of request. As a shortcut, you can leave off **http://** from the URL.

   The HTML markup is imported and displayed in the HTML Text Editor.
3. Install the HTMLText resource by choosing **HTML→Install** and specifying a class and a resource method name.

The HTML markup is imported by copying it into the resource method. Changes made to the original source file after importing will not be reflected in the HTMLText resource.

Note that the HTML Text Editor does not parse the HTML markup. In both the **show HTML** and **show text** modes, the HTML Text Editor shows the raw HTML markup that you imported.

Finally, when you import HTML markup, there may be references to URLs that are relative the document's original host location. There are two ways to make these URLs work:

- You can expand each to a full URL beginning with **http://** by using the **Edit→Make URLs Relative To…** command and entering the document's original protocol, host, and any intermediate directory names.
- You can leave the relative URLs and specify a base URL for the HTML text widget that uses this resource by setting the **Base URL** property for that widget.

Specifying a Base URL

To make all of the URLs in an HTMLText resource share the same base URL, you can embed the base URL in the HTMLText resource's links. The HTML Text Editor automates this process with the **Edit→Make URLs Relative To…** command. Note that this command affects all widgets that use the HTMLText resource.

This can be useful for "fixing" HTML that you have imported from an external file. It does not work if the URLs in your HTMLText resource are already fully-qualified URLs beginning with a protocol.

To embed the base URL in an HTMLText resource:

1. Open the HTML Text Editor.
2. Load the HTMLText resource.

   The links in the HTMLText resource should have relative URLs.
3. Choose the **Edit→Make URLs Relative To…** command.

   When you choose this command, the HTML Text Editor prompts you for the base URL. The base URL is the prefix you want inserted before each relative URL in the current HTMLText resource. The base URL consists of the protocol, host name, and any intermediary directory names for the document.
4. Enter the base URL and click the **OK** button.

   Once you've entered the base URL, the HTML Text Editor searches for all occurrences of HTML tags that use URLs (for ACTION, SRC, and HREF attributes). When it locates an occurrence of one of these tags, if the coresponding URL is relative (not qualified with a host), it will prepend the specified base URL to it.
5. Install the HTMLText resource by choosing **HTML→Install** and specifying a class and a resource method name.

Controlling HTML Encoding

Edit→Encode HTML Characters…

This option only applies when saving or installing the text you are working with. With this option on, the HTML Text Editor encodes any character that is a reserved HTML character when you save or install your work, (bracket, ampersand, quotation marks, and so on).

For example, suppose that your text is:

<enter name here>

The editor will encode the brackets to look like this:

<enter name here>

This allows the client's browser to display the characters rather than trying to interpret this text as an HTML tag.

**Edit→Convert Crs to
…**

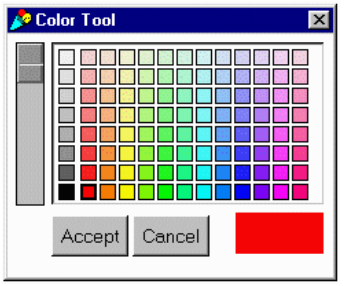
This option only applies when saving or installing the text you are working with. When you toggle this option on, whenever you save or install your work, the HTML Text Editor will convert line breaks that are not paragraph breaks to an HTML break (`<BR>`). The `<BR>` then shows up both when you are showing the text and when you are showing the HTML in the editor.

Adding Color to HTML Text

You can assign colors to sections of HTML text. To add color:

1. Set the HTML Text Editor display to **Show HTML**.
2. Use the mouse to select the text that you want to color.
3. Click the Color button  .

   The HTML Text Editor displays the Color Tool.



4. Click on the color for the selected text and then click the **Accept** button.

5 Install the HTMLText resource by choosing **HTML→Install** and specifying a class and a resource method name.

6

HTML Frames

This section describes VisualWave's support for HTML frames.

VisualWave's HTML frames are compatible with frames as supported by Netscape Navigator 2.0 and later.

This section assumes that you are familiar with Netscape frames. Information about frames can be found in numerous web sites and books.

For official information, see Netscape's *Frames: An Introduction* (http://home.netscape.com/assist/net_sites/frames.html).

Overview

Frames are an extension to HTML that make it possible for documents to divide the browser window into one or more independently-scrollable panes and then to assign a separate document URL to each pane.

VisualWave provides tools for painting a specification for a group of framesets and frames. Each frameset and frame has properties, such as borders. The frame's properties include its initial contents, which can be a window specification for an application model in the VisualWave image or it can be an URL.

To create and edit the specification for a set of frames, use the Frames Editor, which is available from the main **Tools** menu. When the layout is finished, you then install it as a resource in a class whose superclass is CompositeApplication. To start the application, you open the CompositeApplication, which in turn opens the initial contents for each frame.

When working with the HTML frames and the editor, keep in mind that:

- An application that is designed to be displayed in frames cannot be run as a screen application on a workstation; it must always be run from a frames-enabled web browser.
- Framesets and frames are not widgets in the traditional sense; they are divisions of the canvas itself. Thus, every part of the canvas must be contained by a frame.
- Frames may not overlap each other.
- All new frames are created by dividing an existing frame into two equal-sized frames. Frames can then be resized and moved as needed.
- A canvas may contain as many frames as space permits.
- If separate applications are run each in its own frame, each application runs separately, each with its own overhead. If the applications share an intensive process, such as database access, you can reduce the amount of overhead by putting that process in a separate manager layer, and accessing that from the other applications.

Not all browsers support frames, and some users don't like them. To provide a minimal interface for browsers that do not support frames, fill out the **Alternate HTML** property for the frameset.

Dividing a Canvas into Frames

To create an example canvas that is divided into frames:

1 In the Launcher window, choose **Tools→Frames Editor**.

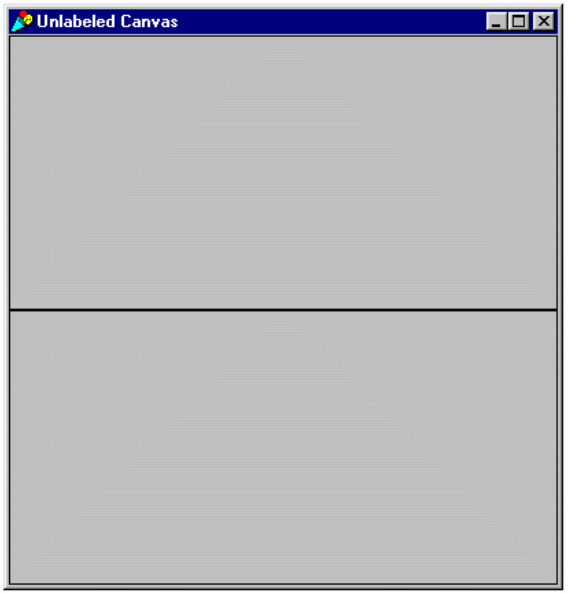
VisualWave opens the Frames Editor. If your image's palettes are set to open with every canvas, VisualWave also opens the Frames Palette. If your image's palettes are set to open on demand, display the Palette by choosing **tools→palette** from the Frames Editor's <Operate> menu.

The initial frameset contains a single frame. Like any HTML frameset, it can be divided into either a set of either horizontal frames (rows) or vertical frames (columns).

2 Divide the canvas into two horizontal frames:

a In the palette, click on the Horizontal Frames button: 

b Click anywhere in the canvas. The Frames Editor divides the canvas into two equally-sized horizontal frames. You always add frames by dividing an existing frame, vertically or horizontally, into two equally-sized frames.



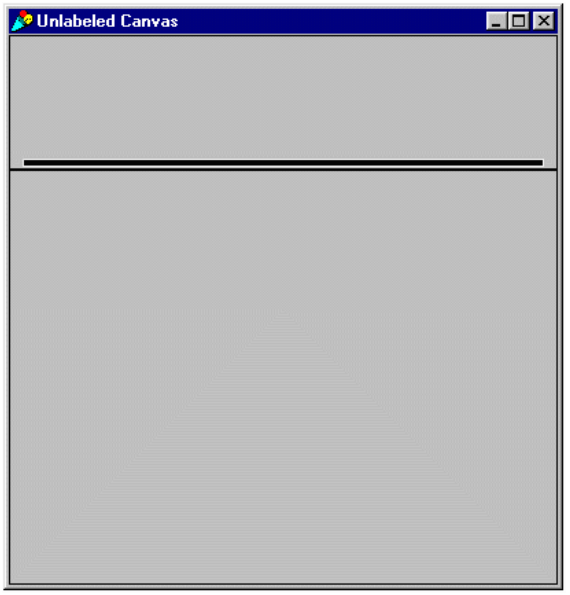
3 Resize the frames so that the top one takes up about 1/4 of the canvas:

a In the canvas, click in the top frame to select it. When it's selected, a black selection handle appears on the bottom edge.

b Position the pointer over the selection handle, hold down the <Select> mouse button, and drag the bottom edge of the frame upward.

c Release the mouse button.

Notice that the bottom frame becomes larger to fill the space. Frame are not like widgets; you do not place them on the canvas and move them around. Instead, frames are divisions of the canvas itself. Every part of the canvas must be in exactly one frame.

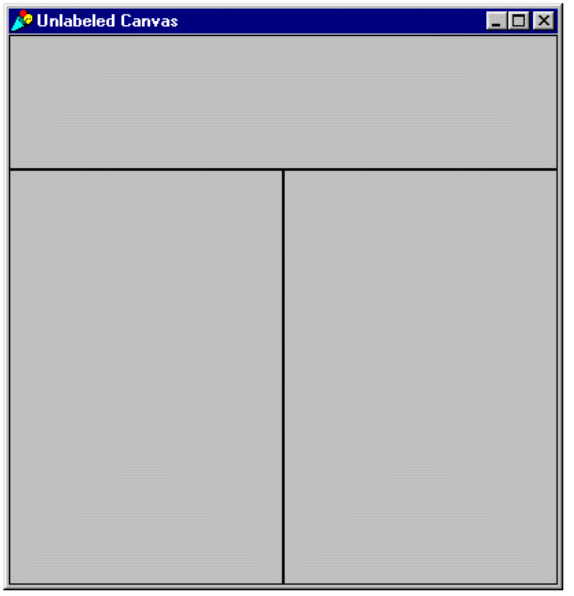


Each of these frames can be further divided into vertical or horizontal frames.

4 Divide the bottom frame into two vertical frames:

a In the palette, click on the Vertical Frames button: 

b Click in the bottom frame. The Frames Editor divides the bottom frame into two equally-sized vertical frames.

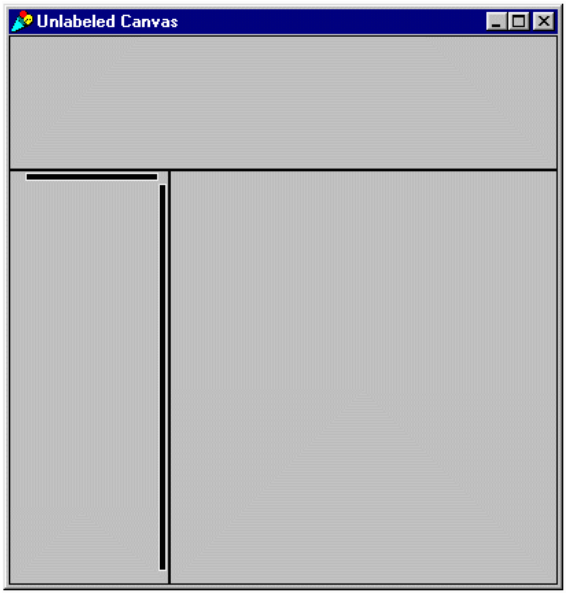


5 Resize the frames so that the bottom left one takes up about 1/4 of the canvas:

a In the canvas, click in the left frame to select it. When it's selected, black selection handles appear on the top and right edges.

b Position the pointer over the right selection handle, hold down the <Select> mouse button, and drag the right edge of the frame to the left.

c Release the mouse button.



At this point, your canvas is divided into two framesets:

- The first frameset divides the canvas into two horizontal frames or rows.
- The bottom frame of the first frameset is itself a frameset, divided into two vertical frames or columns.

To see the HTML that VisualWave generates for this layout, click the **Preview** button in the Palette. VisualWave generates the HTML and writes it to a file named **`preview.htm`**.

The HTML for this layout is:

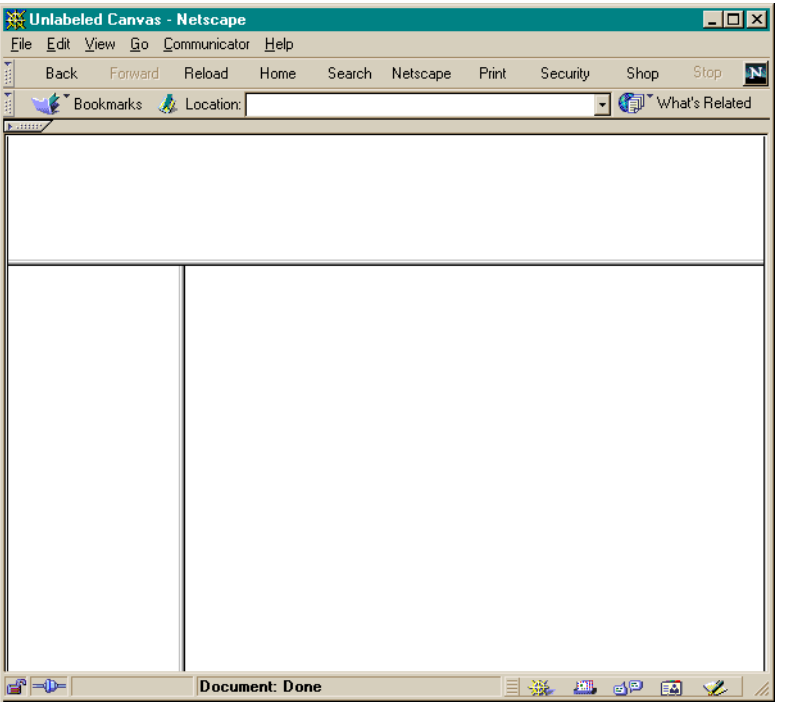
```
<FRAMESET COLS="*" >

<FRAMESET ROWS="25%,*" >
      <FRAME
   SRC="http://localhost:8008/launch/FrameSource@undefinedSpec"
   SCROLLING="Auto" >
      </FRAME>
      <FRAMESET COLS="25%,*" >
         <FRAME
   SRC="http://localhost:8008/launch/FrameSource@undefinedSpec"
   SCROLLING="Auto" >
         </FRAME>
```

```
        <FRAME
    SRC="http://localhost:8008/launch/FrameSource@undefinedSpec"
    SCROLLING="Auto" >
        </FRAME>
      </FRAMESET>
</FRAMESET>

</FRAMESET>
```

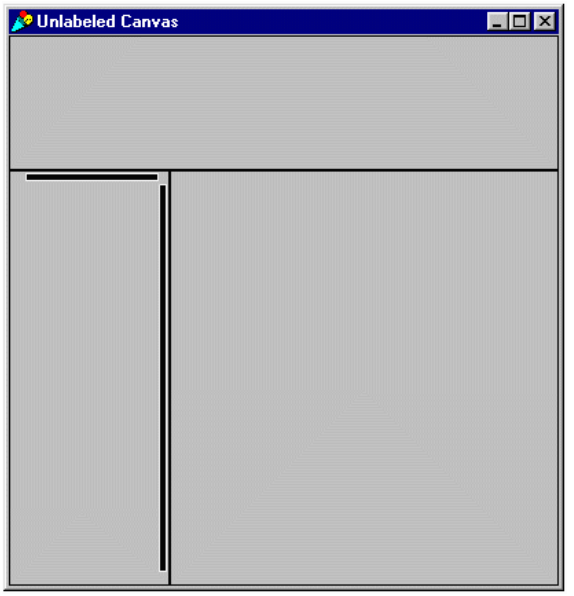
Use your web browser to display the **`preview.htm`** file that VisualWave generated. It should look like this:



Selecting and Deselecting Frames and Framesets

Selecting a Frame

To select a frame, simply click the <Select> mouse button within the boundaries of the frame. Any other frame that was selected becomes deselected. The selected frame is indicated by selection handles along its interior borders. In the picture below, the bottom left frame is selected.



Deselecting a Frame

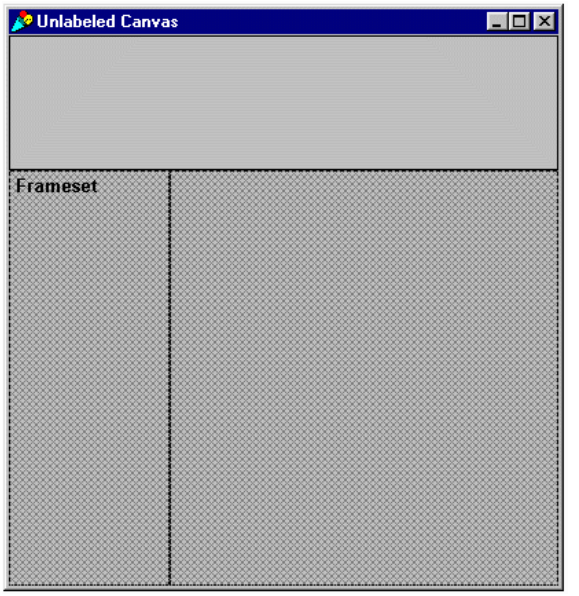
To deselect a frame:

- Click on another frame, *or*
- Hold down the <Shift> key and click the <Select> mouse button

When no frames or framesets are selected, the Properties Tool displays the properties for the window itself.

Selecting a Frameset

You can also select a frameset on the canvas. To select a frameset, hold down the <Alt> key and click the <Select> mouse button within the frameset. The selected frameset is emphasized and “frameset” appears in the upper left corner of the selected area. In the picture below, one of the framesets is selected.



Deselecting a Frameset

To deselect a frameset:

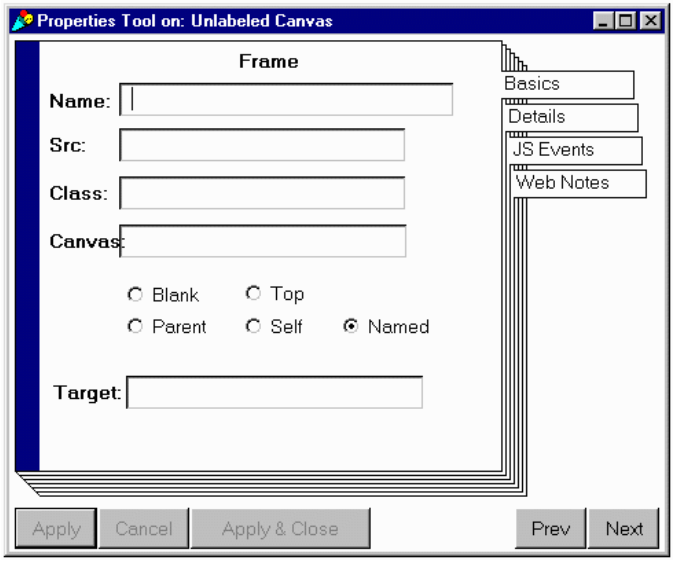
- Click on frame outside the frameset, *or*
- Hold down the <Shift> key and click the <Select> mouse button

When no frames or framesets are selected, the Properties Tool displays the properties for the window itself.

Specifying the Attributes for a Frame

VisualWave supports all of the attributes that can be applied to a frame. They are treated as properties of the frame, similar to the properties for widgets.

To view the properties for a frame, select the frame and click the **Properties…** button on the Palette.



Basics

Name (optional)

A name is necessary if you want to target output into this frame from another frame, an application model, or JavaScript. The name can be any string that is a legal name for an HTML frame.

Src (required)

Every frame must have a source, which can be either:

- An URL, specified in the **Src** field of the Basic properties. That URL will be requested by the user's web browser and must be locatable by that browser, *or*
- A class name and window specification name, specified in the **Class** and **Canvas** fields. The class must be a subclass of ApplicationModel.

The source cannot be any other kind of Smalltalk resource.

Target (optional)

If you want the results of an HTTP `sumbit` from the current frame to be returned in another frame, click the appropriate radio button:

Blank	Returns the result in a new web browser window.
Parent	Returns the result in the frame's parent frameset.
Top	Returns the result in the main browser window. Useful for "breaking out of" a framed application.
Self	Returns the result in the current frame. This has the same effect as not specifying a target at all.
Named	Returns the result in a frame of the name shown in the **Target** input field. That frame must be defined as part of the same frame specification and must have a name. If the input field is blank, no target is specified and the result is shown in the current frame.

The target attribute for a frame sets the base target for the page that is *initially* generated for that frame. Thus, it only affects the *first* application model that is loaded into the frame. For more information about targets, see "Targeting Output" on page 63.

Details

Margin Width (optional)

Specifies the amount of space to leave on at the left and right edges of the frame. May not work properly with some versions of Netscape Navigator.

Margin Height (optional)

Specifies the amount of space to leave on at the top and bottom edges of the frame. May not work properly with some versions of Netscape Navigator.

Resizable (optional)

By default, the user can resize the frames in his browser by selecting the edge and dragging. Select this attribute to generate frames that cannot be resized in the web browser.

Scrolling (optional)

By default, the browser is allowed to add scroll bars when the contents for a frame are larger than the frame. When scroll bars are added, both horizontal and vertical bars are added together. To generate frames that always have scroll bars, choose **On**. To generate frames that never have scroll bars, choose **Off**.

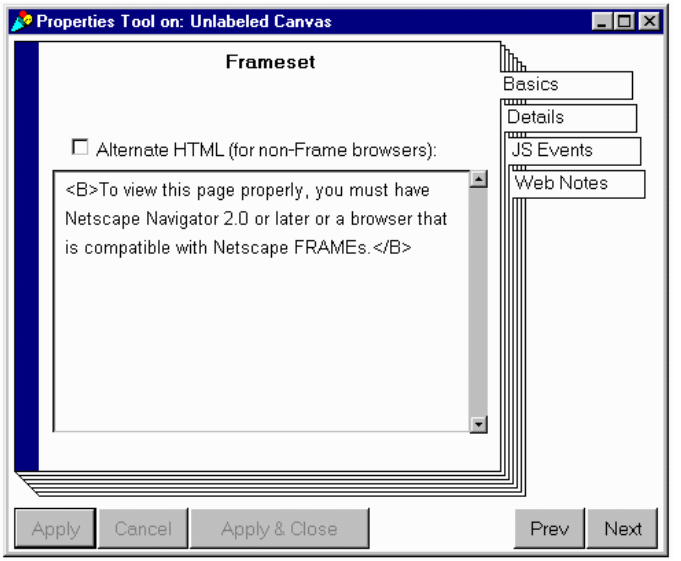
Frame Border

Controls whether or not the frame has a visual border.

Specifying the Attributes for a Frameset

VisualWave supports all of the attributes that can be applied to a frameset. They are treated as properties of the frameset, similar to the properties for widgets.

To view the properties for a frameset, select the frameset and click the **Properties...** button on the Palette.



Basics

Alternate HTML (optional)

When this box is checked, VisualWave includes a **<NOFRAMES>** section in the HTML page that is generated. The **<NOFRAMES>** section contains the HTML shown in the text field below the **Alternate HTML** box.

Web browsers that do *not* support HTML frames display this alternate text. The default text instructs the user to use a browser that supports frames, such as Netscape Navigator 2.0.

Web browsers that support HTML frames ignore this text.

Details

Frame Spacing

Allows you to create additional spacing between frames. The number that you enter is interpreted as a number of pixels.

Frame Border

Allows you to control whether or not there is a visual border around the frameset. Users can resize framesets by dragging the borders displayed in their browsers. By default, a border is included.

JS Events

Framesets support two JavaScript events:

- onLoad
- onUnload

For more information about JavaScript events, see "Client-side JavaScript" on page 71.

Other Operations with Frames

Deleting a Frame

To delete a frame, select it and choose **cut** from the popup <Operate> menu or press the <Delete> key. The Frames Editor removes that frame and extends the other frames in the immediate frameset to fill in the space.

Moving Frames

Frames are not widgets. Thus, they cannot be moved simply by selecting them and dragging them across the canvas.

Rather, frames are divisions of the canvas itself. At every moment, the canvas must be completely covered in frames; no part of the canvas can be outside a frame. Thus, you move frames by switching the location of one frame with another frame. The Frames tool palette contains four buttons that control movement of frames.

Saving a Framed Canvas

To save a frame specification, you *install* it in a class by clicking on the **Install…** button and specifying the name of the class and a name for the window specification that will be created. The class that is created must be a special kind of ApplicationModel, called a CompositeApplication.

Accommodating Browsers that Don't Support Frames

You can specify the message to be displayed in browsers that do not support frames. To do so:

1. Display the frame specification in the Frames Editor
2. Select a frameset by holding down the <Alt> key and clicking on a frameset.
3. Display the frameset's properties.
4. Open the Properties Tool, and select the **Basics** page.
5. On the **Basics** page, check the **Alternate HTML for non-Frame browsers** checkbox.
6. In the text field below the checkbox, there is a default message. You can edit the message to suit your situation. The message can include any standard HTML markup.
7. Apply the changes.
8. Install the canvas.

Examples

Two examples of applications that use frames are included with VisualWave.

ServiceDeskDemo is a very basic customer request and tracking application. ServiceDeskDemo is in the VisualWaveDemos parcel.

CheckbookInterface is a framed version of the Checkbook application. CheckbookInterface is in the VisualWaveDemos parcel.

7

Targeting Output

HTML allows some elements to have a *target* assigned to them. When the user requests a web page, the target determines where the response will be displayed. VisualWave allows you to set the *base target* for a web page, which determines where a submission from any element on that page will display its output. You can also set the target for a label widget that acts as an HTML link.

This section describes VisualWave support for targets.

Allowed Targets

No matter which method you use to set the target, the allowed targets are the same. You can target output to:

Target value	Description
`_self`	The current frame. This is the same as having no target specified. It can be used explicitly when you need to change a target that has already been set.
`frameName`	The frame with the given name. You can specify the name for a frame as part of its properties. Only frames that have a name can be a target. The name can be any string that is a legal name for an HTML frame.
`_parent`	The parent frameset. Use `_parent` if you want the output to take up the current frameset. Using `_parent` removes any other frames that were in that frameset.

Target value	Description
`_top`	The current browser window. Use `_top` if you want the output to take up the entire browser window. Using `_top` removes any other frames and framesets that were in the window.
`_blank`	A new browser window. The current browser window remains on the screen unaltered.

Targeting with the Base Target Property

You can set the base target in the properties for a frame or window (canvas). You should use the **Base Target** property to:

- Set the initial target for a frame.
- Set the initial target for a window.

Setting the Base Target

For a Frame

1. Select the frame.
2. Click the **Properties** button in the HTML Frames Editor's Palette to display the frame's properties.
3. Select the button for the target: **self**, **top**, **parent**, **blank**, or **named**.
4. If you selected the **named** button, enter the frame's name in the **Target** input field.
5. Apply the change.
6. Install the canvas.

For a Window

1. Deselect all the widgets in the window. Hold down the <Shift> key and click the <Select> mouse button on a widget to deselect it.
2. Click the **Properties** button in the Canvas Tool to display the window's properties.
3. Go to the **Web Base** page.
4. In the **Base Target** input field, enter the desired target.
5. Apply the change.
6. Install the canvas.

Targeting with JavaScript

You can use JavaScript to change the base target for the current page. For example, the following JavaScript code sets the page's target to _parent.

```
this.form.target='_parent'
```

You can use JavaScript to change the target based on an event for:

- A window (canvas)
- A frameset
- A widget

This section provides details and an example; for complete information about VisualWave's support for JavaScript, see "Client-side JavaScript" on page 71.

When to Use JavaScript

You should use JavaScript instead of another method to change the target based on:

- A widget event. Only JavaScript allows you to set the target from an arbitary widget. You can set the target for a label widget in the widget's **Web** properties.
- A frameset event. Only JavaScript allows you to set the target from a frameset.
- A window event, other than onLoad. Setting the target in the **Base Target** properties of the window is equivalent to setting the target in the onLoad event of the window.

How to Use JavaScript

Framesets, windows, and some widgets have a **JS Events** properties page. The JavaScript events that are supported by the UI Painter for the selected frameset, window, or widget appear on that page. You can set the target based on any of the events.

1. Select the event that you want to trigger the change of target.
2. Enter the JavaScript code to change the target.
3. Apply the change.
4. Install the canvas.

Note that **`this.form.target`** changes the base target for the entire form. Even when **`this.form.target`** is executed in response to a JavaScript event for a widget, the target is changed for the entire form. Thus, the change affects all submits, from any widgets on that form.

Example: Targeting From a Widget

The most common use of JavaScript is to change the target when a particular widget is clicked. For example, you may want the **Exit** button to target the browser window and remove all of the frames that the application has displayed. In that case, you would set the **Exit** button's **JS Events onClick** event property to be:

```
this.form.target='_top'
```

Targeting from a Window

To set the target at the window level:

- Use the JavaScript **onLoad** event for the window (see "Targeting with JavaScript" on page 65),
  or
- Use the **Base Target** property for the window. The window can be either a regular canvas or a frameset canvas (see "Targeting with the Base Target Property" on page 64).

Targeting from a Frames or Frameset

To set the target for a frame, set the frame's **Target** property (see "Targeting with the Base Target Property" on page 64 above).

> **Note:** The target attribute for a frame sets the base target for the page that is *initially* generated for that frame. Thus, it only affects the *first* application model that is loaded into the frame.

To set the target for a frameset based on:

- Loading or unloading of a frameset, use JavaScript (see "Targeting with JavaScript" on page 65),
  or
- Loading the main frameset of a frameset canvas, use the **Base Target** property for the window (see "Targeting with the Base Target Property" on page 64).

Targeting from a Widget

To change the target based on interaction with a widget, use JavaScript (see "Targeting with JavaScript" on page 65).

Targeting from a Label Widget

Label widgets can be used as HTML links. When they are, you can assign a target to them. Clicking on the label takes the user to the assigned URL and displays the results in the assigned target frame.

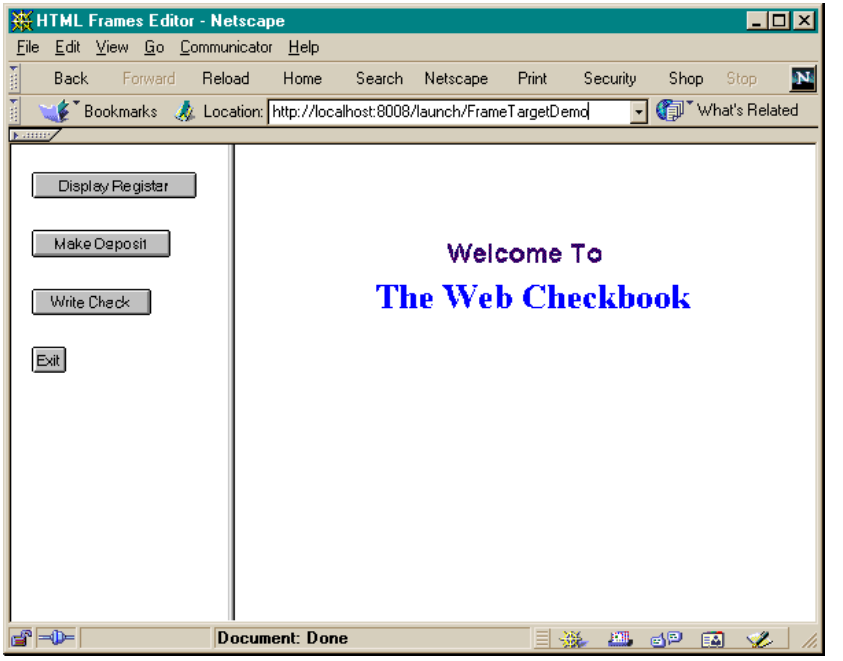
To set the target for a label widget:

1. Place the label widget on the canvas and display its properties.
2. Specify the text that you want to appear on the HTML page displayed to the user:
   a. Choose the **Basics** page.
   b. In the **Label** property, enter the desired text.
   c. Click the **Apply** button.
3. Specify the link information, which determines which location will be displayed when the user clicks on the label:
   a. Choose the **Web** page.
   b. Select the anchor type **HREF**.
   c. Select the **Protocol** HTTP.
   d. In the **Enter an URL for the link** field, enter a complete URL including the HTTP protocol.
   e. Click the **Apply** button.
4. Specify the target information:
   a. On the **Web** page, select a predefined target (blank, top, parent, or self) or select the **Named** option.
   b. If you selected the **Named** option, enter the name of a frame.
   c. Click the **Apply** button.
5. Install the canvas.

Examples: Targetting Output

Example 1: Frame and Widget

Suppose that your application has two frames. The left frame contains a series of buttons. All of the buttons except one should cause VisualWave to target the answering page to the right frame. The **Exit** button should cause VisualWave to target the answering page to the parent frameset:



You can implement this by:

- Setting the left frame's **Target** property to be the name of the right frame.

  It makes sense to set the target at the frame level because the contents of this frame do not change. Once the initial control panel is displayed, it remains in the left frame. If the left frame were ever updated, the new contents would not inherit the target.

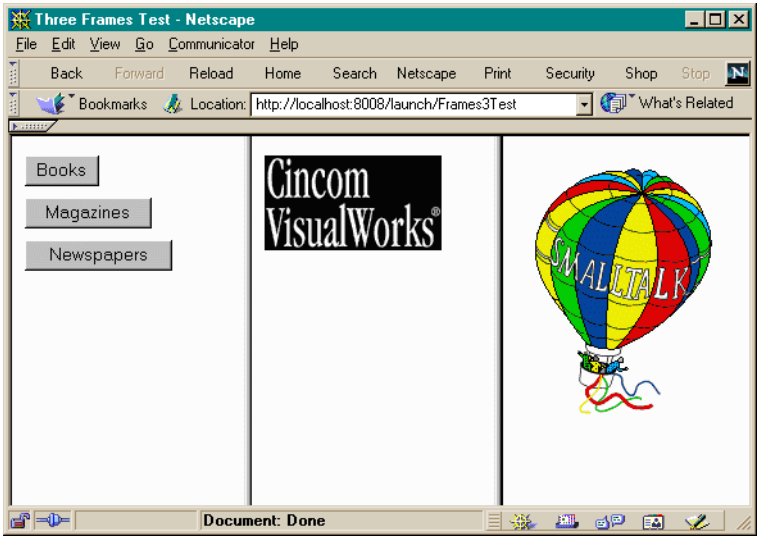
- Setting the **Exit** button's **onClick** JavaScript event property to change the target to **_parent**.

The only way to target a particular widget's response to a different location is to use a JavaScript event. Note that JavaScript changes the target for the entire form. The new target affects submits from all of the other buttons as well. In the case of the **Exit** button, that's okay; the user will not be clicking another button.

Example 2: Frame and Window

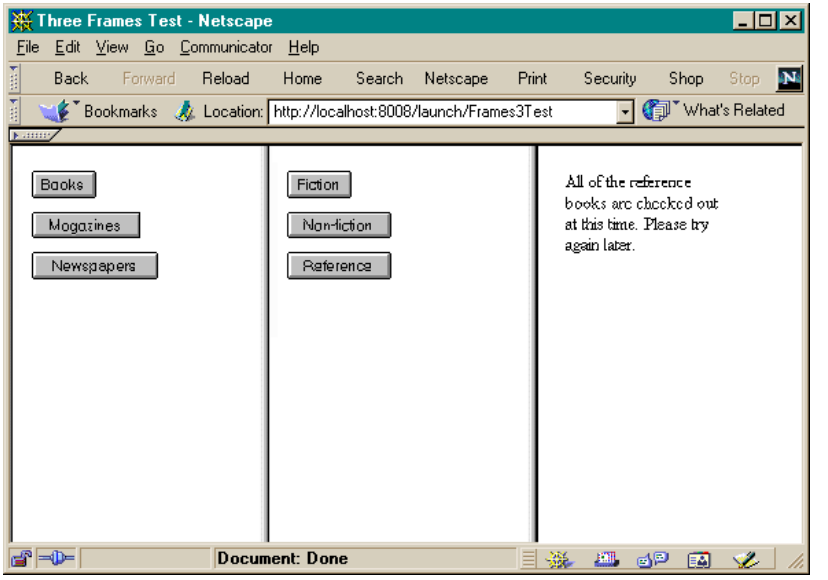
Suppose that your application has three frames. When the application first starts:

- The left frame has a set of control buttons or options
- The middle frame has a logo
- The right frame has a logo



When the user clicks on one of the controls in the left frame, the middle frame is updated so that it contains another set of controls. When the user clicks on one of the controls in the middle frame, the right frame is updated. Think of the three frames as an outline, each displaying more details depending on what is selected in the frame to its left.

In the example below, when the user clicked the **Books** button, the middle frame was updated to show the choices for kinds of books. When the user clicked the **Reference** button, the right frame was updated to explain that all the reference books are checked out.



You can implement this by:

- Setting the left frame's **Target** property to be the name of the middle frame.

  It makes sense to set the target at the frame level because the contents of this frame do not change.

- Setting the BookKinds canvas's **Base Target** property to be the name of the right frame.

  You cannot set the target in the middle frame because this canvas is not initially displayed in the middle frame. When it is displayed, it overwrites all of the previous frame contents.

  You cannot set the target in the middle frame because this canvas is not initially displayed in the middle frame. When it is displayed, it overwrites all of the previous frame contents, including the target setting. Specifying the target at the window level ensures that the target is set when the window is loaded.

Note that both the middle and right frames must have names in the specification for the frameset.

8

Client-side JavaScript

VisualWave's JavaScript support is compatible with JavaScript as supported by Netscape Navigator (3.0 and higher), and is compatible with Microsoft Internet Explorer (4.0) to the extent that it supports the Netscape definition.

This chapter assumes that you are familiar with JavaScript. Information about JavaScript can be found in numerous web sites and printed books. For official information, see Netscape's *JavaScript Authoring Guide* (http://home.netscape.com/eng/mozilla/gold/handbook/).

Why Use JavaScript?

All application logic in VisualWave is implemented on the server side. This makes it difficult or impossible to do such things as pop-up help, submitting when a radio button is selected, or client-side validation of input fields.

JavaScript fills this gap by letting you define event handlers for events such as clicking on a form element, moving the input focus out of a field, or moving the mouse pointer over a link. JavaScript support in VisualWave allows you to associate JavaScript code with events for a variety of VisualWave widgets. Those widgets and the supported events are described below.

VisualWave uses JavaScript to submit an HTML form when a selection is made in a list or by selecting a Check Box, Radio Button, or Menu Button.

VisualWave does not automatically generate any JavaScript code to do field validation, nor does it attempt to translate any Smalltalk validation methods.

Supported Widgets and Events

JavaScript defines the events that are recognized for each HTML tag. You can associate JavaScript code (JavaScript function calls and/or other JavaScript code) with a recognized event for VisualWave widgets as listed in the following table.

VisualWave Widget	HTML Tag	Recognized Events	
		Netscape 2.0	Netscape 3.0
Window*	form, body	onSubmit, onLoad, onUnload	onSubmit, onLoad, onUnload, onBlur, onFocus, onError
Frameset	frameset	onLoad, onUnload	onLoad, onUnload, onBlur, onFocus
Frame	frame		onBlur, onFocus
ActionButton	button	onClick	onClick
ActionButton**	image	none	none
Check Box	check box	onClick	onClick
Radio Button	radio button	onClick	onClick
Label (plain)	none	none	none
Label (link)	link	onClick, onMouseover	onClick, onMouseover, onMouseout
Label (link with label as image)	link and image	onClick, onMouseover	onClick, onMouseover, onMouseout, onLoad, onAbort, onError
Input Field	input field (text field)	onFocus, onBlur, onChange, onSelect	onFocus, onBlur, onChange, onSelect
Text Editor	text area	onFocus, onBlur, onChange, onSelect	onFocus, onBlur, onChange, onSelect
Menu Button	select	onFocus, onBlur, onChange	onFocus, onBlur, onChange
List	select	onFocus, onBlur, onChange	onFocus, onBlur, onChange

* The onSubmit event is actually associated with the FORM section of an HTML document. The onLoad, onUnload, onError, onBlur, and onFocus events are associated with the BODY section. Because the mappings are unambiguous and neither the FORM nor BODY sections are surfaced as widgets in the canvas, these events are mapped to the window widget.

** For buttons with **Enforce Boundary with GIF** enabled.

JavaScript Properties

JavaScript Properties for Windows

The window itself has three JavaScript properties pages in the UI Painter Properties tool:

JS Src allows you to reference JavaScript functions that are in an external file. External files are referenced by URL. This page also allows you to set the text that is displayed in web browsers that do not support JavaScript.

JS Functions allows you to define functions that can be called by the widgets on the window or its subcanvases. Functions defined here are included in the **<HEAD>** section of the page so that they are guaranteed to be downloaded before they are called.

JS Events allows you to associate JavaScript code with events that apply to the window, such as loading and unloading.

JavaScript Properties for Widgets

VisualWorks widgets that get translated to HTML tags with recognized events (see the list above) have a **JS Events** properties page. This page contains a list of supported events and a text pane where you can type in arbitrary JavaScript code.

When you click on an event handler name in the **JavaScript Event** list, its associated event handler code (if any) appears in the JavaScript text pane. You can associate each recognized event with an arbitrary amount of JavaScript code. You can enclose strings in either single or double quotation marks, but do not mix single and double quotation marks in a JavaScript code segment (see “Single vs. Double Quotation Marks” on page 77). You can refer to other fields on a page by name (see “Accessing Named Fields” on page 76).

> **Tip:** You can test JavaScript code in a Netscape browser by choosing **File-Open Location** and entering **javascript:** for the location. Netscape then opens a window in which you can enter and evaluate JavaScript code.

After you've entered code, click **Apply** to save it.

Once code has been applied, the event handler name appears in bold so you can immediately tell which event handlers are defined. To "undefine" an event, delete all code (whitespace will be ignored in this case) and apply the change; the event handler name will return to the normal font.

Although you can define functions in a widget's JavaScript page, normally you'll define the functions elsewhere and call them from the widget event handlers. Normally, functions are defined on the **JS Functions** properties page for the widget's window.

Labels are transformed into links (with click and mouseover events) or plain text (with no recognized events). The **JS Events** page is present for all labels, whether they have events or not.

A widget with **Label as Image** checked turns into a MappedClickWidget. The corresponding field type does not support any JavaScript events, so the onClick event handler is ignored when the HTML is generated.

Using JavaScript

Browser Support of JavaScript

Keep in mind that not all browsers support JavaScript. VisualWave encloses JavaScript code in old-style HTML comments so the code will be ignored by non-JavaScript enabled browsers. You can specify the message to be displayed in browsers that do not support JavaScript. To do so:

1. Display the canvas and its properties.
2. In the Properties Tool, display the **JS Src** page.
3. On the **JS Src** page, check the **Alternate HTML for non-JavaScript browsers** checkbox.
4. In the text field below the checkbox, there is a default message. You can edit the message to suit your situation. The message can include any standard HTML markup.

5 Apply the changes.

6 Install the canvas.

Note that even browsers that do support JavaScript are not necessarily equal. Unless you know exactly which browsers your audience will use and/or until there is a JavaScript standard definition, you may want to minimize the reliance on JavaScript.

Submit on Selection

If you check the **Submit form when selection is made** checkbox on the **Web** page for a widget, the following code is appended to the appropriate event handler:

```
/* The following was added when the **Submit form
when selection is made** property was enabled. */

this.form.submitController.value=this.name;
this.form.submit()
```

For List Views and Menu Buttons, the code is added to the onChange event handler. For Check Boxes and Radio Buttons, the code is added to the onClick event handler.

You can add other JavaScript code around this code. If you delete the code, VisualWave automatically unchecks the **Submit form when selection is made** checkbox. You can, however, delete the comment without confusing the checkbox.

Submit from an Artibrary Widget

Any JavaScript code can trigger a submit. If you want to submit from an arbitrary field or event handler, you can include code in the event handler. The code you include is similar to that generated by VisualWave when you click the **Submit form when selection is made** checkbox for a widget. The first line fills in a value for a hidden variable that tells VisualWave which field caused the submit. The second line performs the submit.

Be careful with anchors. A hypertext link is not a form element, so you'll have to modify this code or you'll see the browser error **`"Form has no properties"`** if you try to use this code in a label/anchor.

Accessing Named Fields

JavaScript allows you to access fields by the HTML name. For example, if you write HTML that includes:

```
NAME='statefield'
```

you can refer to this field by its name **statefield**. VisualWave's JavaScript support provides access to named fields by using their IDs from the properties page.

For non-Smalltalk objects such as JavaApplets, you can use the ID field on the properties page to access an entity by name. For example:

```
document.blinkApplet.stop()
```

For widgets that have live Smalltalk objects associated with them, you must refer to this field in a script using that name prefixed with a **$**. For example:

```
$statefield.value = 'TX'
```

sets the value of the state field to the string "TX", assuming that you have properly qualified the name. To set the value of an input field from an onLoad event handler at the window level, you need to use something like:

```
document.forms[0].$statefield.value = 'tx'
```

Assuming that you've defined a function named toHex, you could convert the new value of an input field to hex format and display it in a second input field with the following:

```
this.form.$hexfield.value = tohex(this.value)
```

VisualWave dynamically generates names for fields based on their keys and phases to keep the web browser objects synchronized with the Smalltalk objects. When generating the HTML, VisualWave replaces the user-defined ID-based names with the real key/phase names. This is why you may see an error message from Netscape referring to a field with the name **v2345w5** instead of **$statefield**. To see the mapping, open an inspector on the WebSession and examine the dictionary in WebRequest.

Subcanvas Support

As far as JavaScript is concerned, all subcanvases disappear and all fields are at the same level. There is a distinction between a local and a global submit, but that's handled on the Smalltalk side. The good news is that this means that you can refer to a named field from any event handler as if it were implemented on the top level page. The bad news is that you must be careful to avoid duplicating ids or you will get strange results.

If you define functions at the window level for a window specification that is later used as a subcanvas, these functions will be appended to those of the main window and included in the **<HEAD>** section. Be careful to avoid duplicating function names in a subcanvas: in Microsoft Internet Explorer, duplicates cause an error on loading.

Window event handlers for the subcanvas, if any, are ignored.

Single vs. Double Quotation Marks

In the HTML generated by VisualWave, the event handler code needs to be enclosed in single or double quotes. You can use either, but not both, within the same event handler code.

If VisualWave first finds a single quotation mark in your JavaScript code (even if it's in a comment), it generates double quotation marks around the event handler code and vice versa. Therefore, if you enter the following JavaScript code for the `onLoad` event,

```
alert('hello')
```

the following HTML is generated:

```
onload="alert('hello')"
```

If, however, you use both:

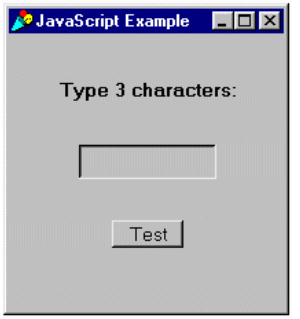
```
alert('single quotes around a string');
  alert("double quotes")
```

VisualWave does not massage the code, so you get a syntax error when you load the page.

Example: Validating Input with JavaScript

JavaScript is often used to validate input on the client side (in the web browser) before submitting the input to the server. In this example, you'll learn how to use VisualWave's JavaScript support to verify that the user has entered exactly three characters in an input field.

1. Start VisualWave.
2. Create a canvas with label, an input field, and an action button as shown here:

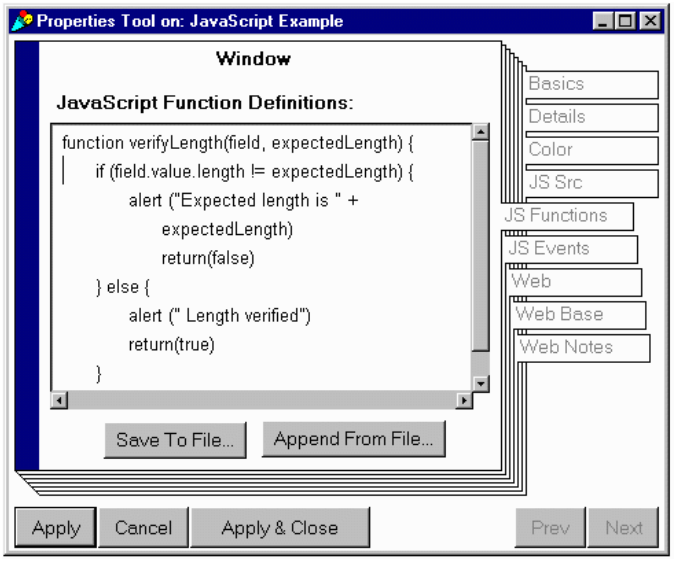


3 Define the JavaScript functions for the canvas:

a Display the properties for the window.

b On the **JS Functions** page, add this JavaScript function:

```
function verifyLength(field, expectedLength) {
      if (field.value.length != expectedLength) {
          alert ("Expected length is " +
              expectedLength)
              return(false)
      } else {
          alert ("Length verified")
          return(true)
      }
}
```



c Apply the change.

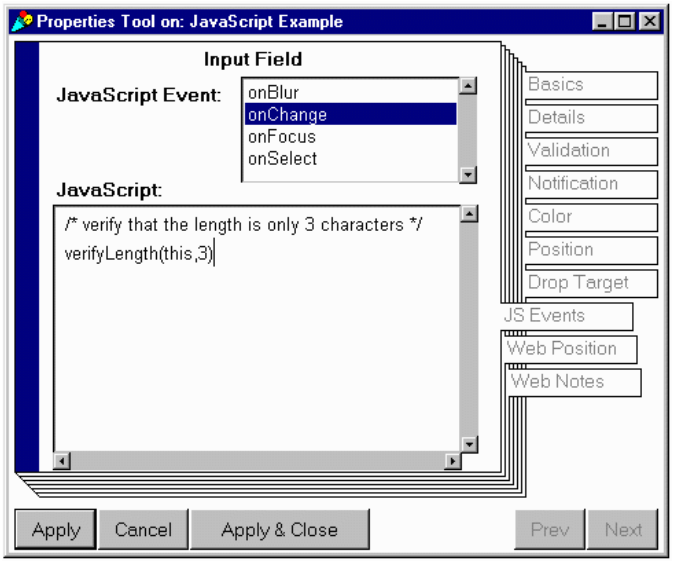
4 Define the event handler for the input field:

a Display the properties for the input field.

b On the **JS Events** page, select the **onChange** event.

c Enter the JavaScript code shown here:

```
/* verify that the length is only 3 characters */
verifyLength(this,3)
```



d Apply the change.

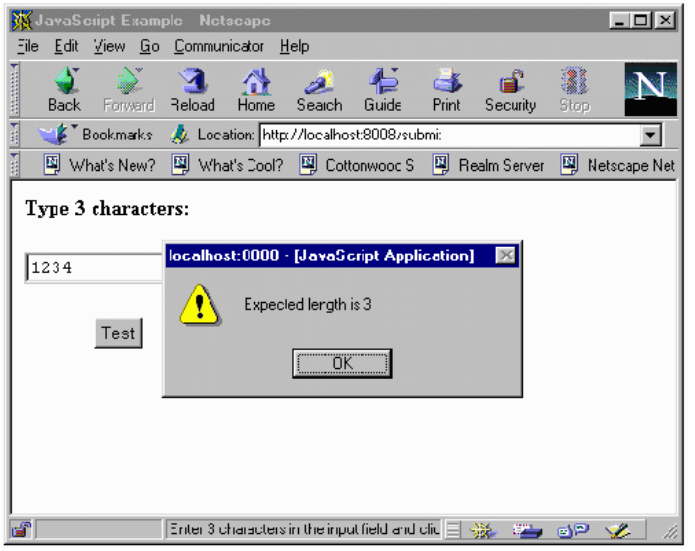
5 To change the status area (the area at the bottom of the Netscape browser) to contain a helpful message when the cursor enters the input field, define the event handler for the **onFocus** event and Apply the change:

```
window.status="Enter 3 characters in the input
field and click the Test button"; return true
```

6 Install the canvas on class JavaScriptExample and selector windowSpec. When prompted, create the class with the following attributes:

Name	JavaScriptExample
Category	JavaScriptDemos
Superclass	ApplicationModel

7 When you run this example in a Netscape browser and type in more than three characters, Netscape displays an alert:



Questions and Answers

Q: I'm using Internet Explorer. Some of the JavaScript examples work, but many do not. What's wrong?
A: Microsoft has stated that they will support JavaScript, but their support seems to lag behind Netscape quite a bit. Test your application carefully with all browsers and platforms you expect your customers to use before relying on a particular JavaScript feature. Coming soon: access to the NOSCRIPT tag to print out a message for browsers that are not JavaScript-enabled.

Q: Can I add my own new JavaScript events? For example, I want to validate each key stroke, so I need an **`#onKeyPress`** event.
A: No. Users cannot define new JavaScript events, and JavaScript as supported by Netscape 4.0 does not recognize keystroke events. It does trigger the **`#onChange`** event when an input field's contents have changed and the input field loses focus but at the present time, you are not notified of individual key presses.

Q: I'm using Netscape 2.02. Why doesn't the **`window.defaultStatus`** property or some of the events such as **`onSelect`** work correctly?
A: These are among some of the bugs that have been fixed in Netscape 3.0 and higher.

Q: Can I use JavaScript to control where the output is sent when a button is clicked?
A: Yes. See the information about targets in "Targeting Output" on page 63

Additional Examples

VisualWave comes with a few examples that use JavaScript.

JADemo shows how you can use JavaScript to call Java methods. It uses the Blink class from the JDK demo directory. JADemo is in the VisualWaveDemos parcel.

Verify uses JavaScript to verify data entered in input fields. Its docExampleSpec matches the tutorial described above. Verify is in the VisualWaveDemos parcel.

CheckbookInterface is a version of the Checkbook application that is described in *Getting Started with VisualWave*. It uses JavaScript to display status information at the bottom of the browser window to verify input before submitting the form to VisualWave, and to change the target when the Exit button is clicked. CheckbookInterface is in the VisualWaveDemos parcel.

9

VRML Widget

This section describes VisualWave's support for the Virtual Reality Modeling Language (VRML) and VRML worlds.

The following discussion presupposes that you are familiar with VRML. The *VRML Repository* (http://www.sdsc.edu/vrml) contains a complete set of links to hardware and software for viewing virtual worlds, collections of worlds, and documentation.

Prerequisites

Both you and your application's users must have a web browser that supports VRML:

- Microsoft Internet Explorer version 3.0 or higher with a VRML 1.0-compliant plug-in 3D browser.
- Netscape Navigator version 3.0 or higher with a VRML 1.0-compliant plug-in 3D browser.

Testing VRML with VisualWave

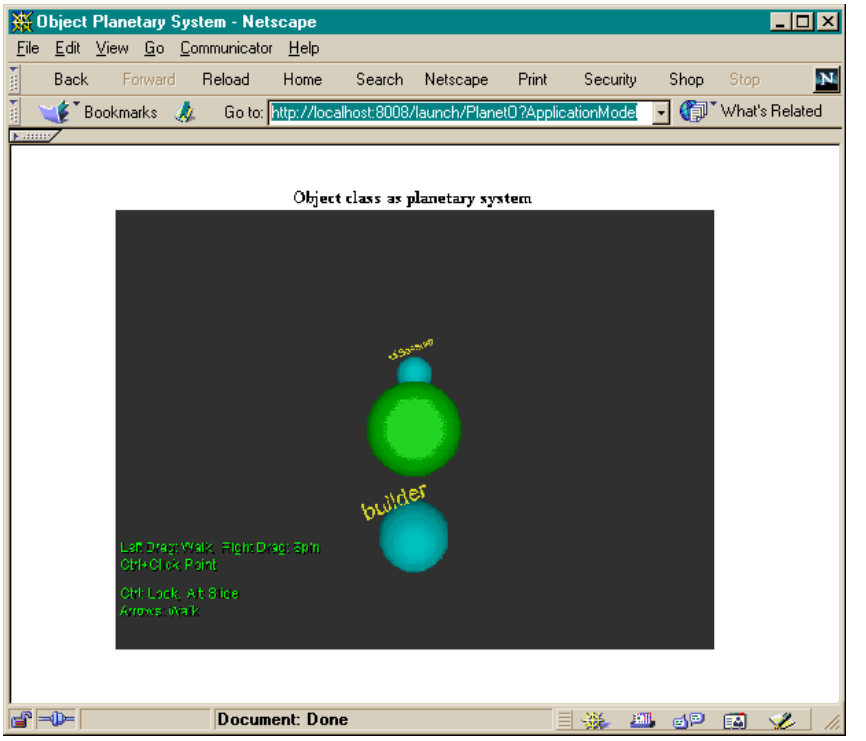
To test your installation:

1. Launch a web browser with the VRML plug-in installed.
2. Start VisualWave.
3. From the Server Console, start a TinyHttpServer. In this example, the TinyHttpServer has the host **localhost** and the port number **8008**.
4. Load the VisualWaveDemos parcel.

5 In the web browser, request the following URL:

http://localhost:8008/launch/Planet0?ApplicationModel

A web page such as the following should be displayed:



Your VRML area may contain different navigational instructions or menus, but the center three planets should be the same. If you do not see a VRML area at all, there is a problem with your setup. Refer to the VRML plug-in vendor's documentation for setup and navigation information.

Tutorial: A Simple VRML World

In this example, you will build a simple VRML world, consisting of a single cylinder. You will learn how to:

- Incorporate VRML into a canvas
- Set up the VRML application model class
- Create the VRML scene

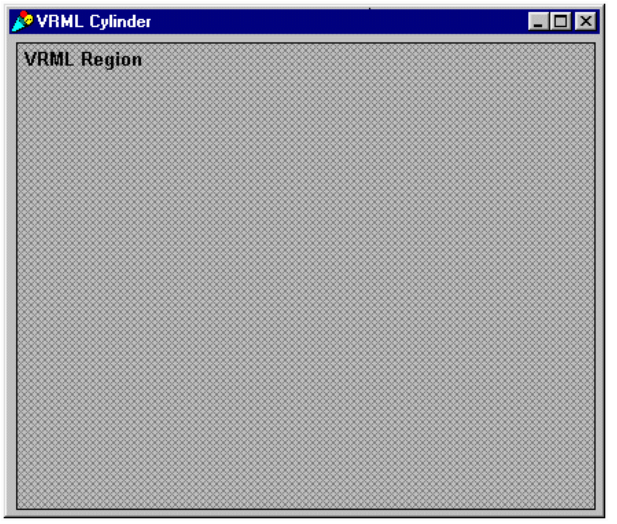
To build a VisualWave application that includes a VRML scene:

1 Open a new canvas by choosing **Tools→New Canvas** in the Launcher window.

2 Resize the canvas to provide ample room for the VRML scene. Store the new canvas size by choosing **Layout→Window→Preferred Size** from the Canvas Tool.

3 Select the VRML Region widget from the canvas: 

4 Place the VRML Region on the canvas and resize it to fill most of the canvas.



5 Set the VRML Region's **Aspect** property to #scene and apply the change.

6 Set the window's **Label** property to **VRML Cylinder** and apply the change.

7 Install the canvas in class VRMLCylinderExample under the selector windowSpec.

8 When prompted to create the VRMLCylinderExample class, enter the following values:

Category	VRMLDemos
Superclass	VRMLAppModel

Every canvas that contains a VRML Region must be a subclass of VRMLAppModel.

9 Close the painting tools.

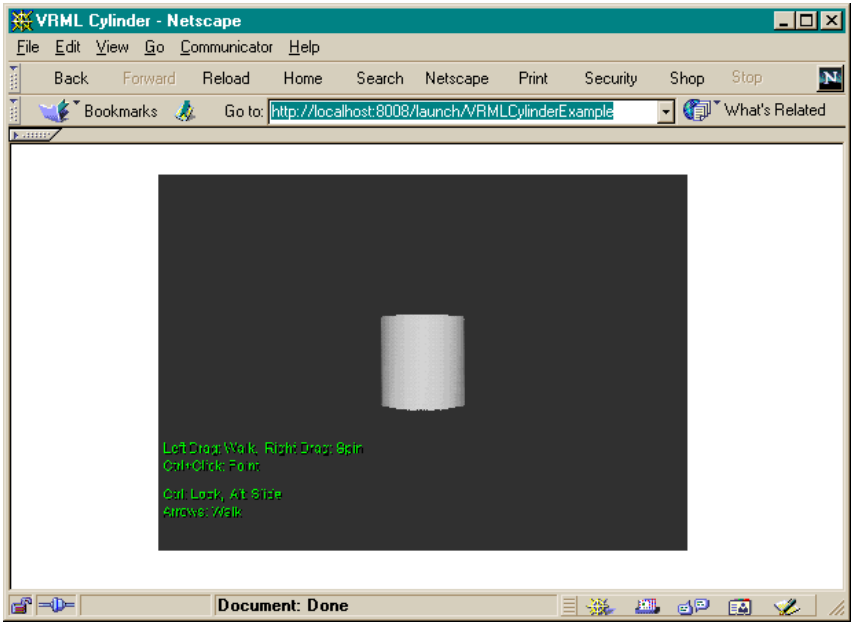
10 Using the System Browser, display the VRMLCylinderExample class. At this point it has only one class method that returns the canvas you painted. There are no variables or instance methods. The class does, however, inherit a scene instance variable and method from its superclass VRMLAppModel.

11 Create the following instance method in a protocol named private.

```
buildVRMLScene

    | root |
    scene := VRMLScene new.
    root := scene root.
    root add: (PerspectiveCamera new positionX: 0 y: 0 z: 8).
    root add: (Cylinder radius: 1 height: 2)
```

12 Test your example by running VRMLCylinderExample from a VRML-enabled web browser. The result should be a VRML space with a cylinder in it:



13 Experiment with this simple VRML scene:

- Practice navigating around the scene.
- Change the PerspectiveCamera coordinates in the buildVRMLScene

method and restart the application.

- Change the Cylinder's radius and height in the buildVRMLScene method and restart the application.
- Try adding other elements to the scene. Classes for the VRML basic shapes can be found the category VRMLShapes.

For more complex VRML examples, see "Additional VRML Examples" on page 87.

Exploring the VRML Classes

Basic VRML Functionality

VisualWave includes a wide number of classes that allow you to program VRML 1.0 worlds in Smalltalk. All classes are in categories whose names begin with "VRML". Some categories that you may want to explore first are:

- VRMLShapes
- VRMLGroups
- VRMLGeometry
- VRMLText

Reserved Classes

The following categories contain classes that are used by VisualWave to generate VRML worlds and should *not* be modified:

- VRMLDev
- VRMLApplications

Additional VRML Examples

VisualWave comes with a few examples that use VRML:

- **HelloVRMLWorld** much like the VRMLCylinderExample described in the tutorial above.
- **DiskSpaceUsage** demonstrates the ability to dynamically generate scenes based on domain information gathered prior to rendering. To run Planet0, you must use an URL of the form:

http://host:port/launch/PlanetO?ClassName

PlanetO also demonstrates the ability to dynamically generate scenes.

The VRML examples are all in the VisualWaveDemos parcel.

Implementation Notes

World Size

When preparing world (`.wrl`) files, you should keep them well below 1Meg in size for VRML browsers to work properly. VRML browsers receiving `.wrl` content larger than this size have been known to lock up the operating system in worst case.

10

Java Applet

This section describes VisualWave's support for Java applets. It includes these topics:

Using Java Applets

VisualWorks allows you to include Java applets and to communicate with them using JavaScript. To add a Java applet to a canvas:

1. Display a UI Painter canvas for editing.
2. From the Palette, choose the Java applet widget  and place it on the canvas.
3. Set the properties for the widget (see table below).
4. Install the canvas.

Java Applet Properties

Page	Property	Required	Description
Basics	ID	No	The name of this component.
	Applet	Yes	The name of the Java applet's object code (`.class`) file. This file must be in the directory named in the Code Base property.
	Code Base	Yes	An URL that points to the directory containing the Java `class` file(s). The Java code must be accessible from some web server.
	Alternate HTML	No	When checked, the HTML in the text field below is shown in web browsers that cannot run Java applets.
Details	Can Access JavaScript	No	When checked, it allows your applet code to call JavaScript code. This property sets the `MAYSCRIPT` attribute of the `APPLET` tag.

Parameters	Parameters	No	Enables you to pass information to the Java applet. Parameters should be entered one per line, in the format:
			`name=value`
			`name2=value2`
			`name3=value3`

Setting Properties Programmatically

Parameters

In addition to being set from the Properties Tool, Java parameters can be set programmatically in the instance of HTMLJavaApplet that represents the Java applet widget. For example, use the following code fragment to add a new parameter and set its value:

```
postOpenWith: aBuilder
    "Add a parameter named 'BackToWaveUrl' with a value
    that contains an url that represents this page with
    an additional jabber component."

    self executeForWebOnly:
        [(self builder componentAt: #wavebackRegion)
        widget htmlEntity add:
            (HTMLParameter new
                name: 'BackToWaveUrl';
                value: (self builder window urlForRefresh ,
                    '&jabber='))].
```

Other Properties

You also can set HTML attributes of an applet that aren't surfaced in the Properties Tool. For example, use the following code fragment to set the vspace parameter of an HTMLInsertObject:

```
postBuildWith: aBuilder
    | webJKGTable |
    webJKGTable := aBuilder componentAt: #table.
    webJKGTable htmlEntity vspace: 8
```

Examples: Using Java Applets

VisualWorks comes with two examples that use Java applets:

StarFieldTest embeds the applet from the BounceItem class found in http://java.sun.com/applets/applets/BouncingHeads. BounceTest is in the VisualWorksDemos parcel.

JADemo shows how JavaScript can call Java methods using the Blink class from the Java Developers Kit. You can download that from http://java.sun.com/products/JDK/. JADemo is in the VisualWorksDemos parcel. See the JADemo class comment for configuration information.

11

Bookmarks

Overview

In VisualWave, you can assign aliases for URLs and then reference those aliases in your applications. Each alias and URL pair is called a *bookmark*. In VisualWave, bookmarks exist in named sets. Each *bookmark set* can have any number of bookmarks, organized into any number of groups or *categories*.

Bookmark sets can be saved to and read from files, enabling you to move bookmarks to the VisualWave Server or other development environments when you move your application. Using bookmarks makes it easier to move your applications from the development environment to VisualWave Server because you don't have to replicate the directory structure. It also makes it easier to update an URL that is used in multiple applications or in multiple places in the same application; you can simply update the bookmark.

URL Cache

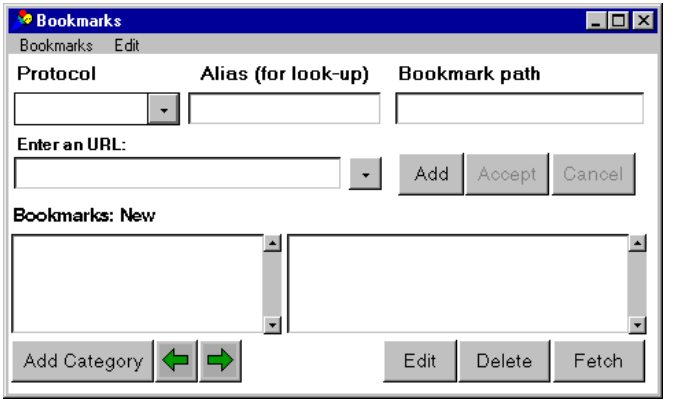
The VisualWave image keeps a cache of URLs. This cache is used to populate the:

- **Enter an URL** menu of the Bookmark Manager
- **Enter an URL for the link (HREF)** menu of the HTML Text Editor.
- **Background Pattern** menu of the Web properties for a window.
- **Background Sound** menu of the Web properties for a window.
- **Enter an URL for the link (HREF)** menu of the Web properties for a label.

You can control the number and kinds of URLs that are cached by setting preferences in the Bookmark Manager.

Bookmark Manager

You create and edit sets of bookmarks using the Bookmark Manager. To open the Bookmark Manager, choose **Tools → Bookmark Manager** in the Launcher window.



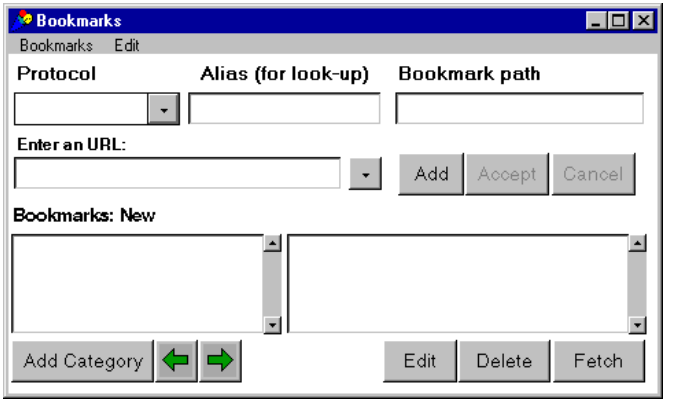
When you open the Bookmark Manager, it displays an empty set of bookmarks. The set has the initial name of "**New**." You can change the name when you save the bookmark set.

For a complete functional description of the Bookmark Manager, see "Bookmark Manager" on page 147 in Appendix A.

Example: Using the Bookmark Manager

Creating Bookmarks

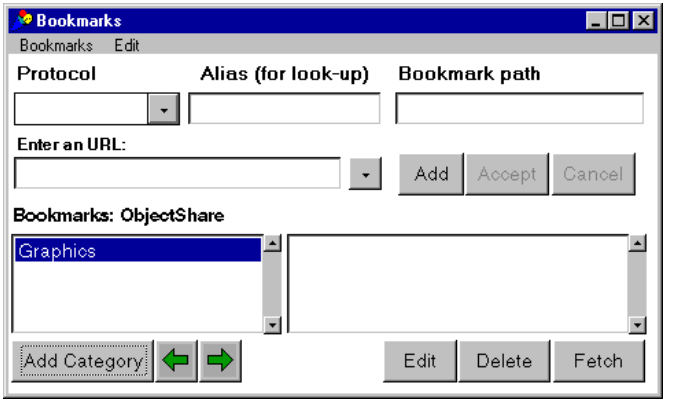
1 Open the Bookmark Manager by choosing **Tools→Bookmark Manager** in the main window.



Initially, the Bookmark Manager displays a new, empty set of bookmarks with the name **New**. In this tool you can load, create or edit, and save bookmarks.

2 Create a bookmark set by choosing **Bookmarks→New**. When prompted, enter a name for the new bookmark set. For this example, use the name **Cincom**.

3 Click the **Add Category** button and enter the name for the new category in the prompter. For this example, enter the name **Graphics**.



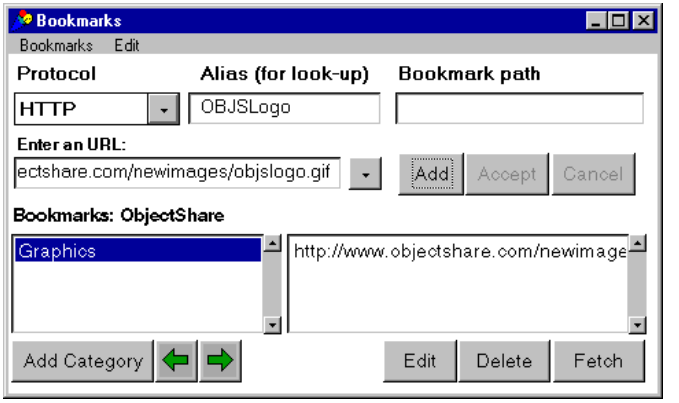
4 For the Protocol, select **HTTP**. When you select HTTP, **http://** is automatically entered into the **Enter an URL** field.

5 In the **Enter an URL** field, complete the URL. The URL may reference a server within the VisualWave image or an external HTTP server. It must be a full URL beginning with the protocol. For this example, enter the URL **http://www.cincom.com/newimages/vwscrstkr.jpg**.

Note: If the Cincom web site changes this image file might no longer be available. You may substitute the URL for any graphic on the web.

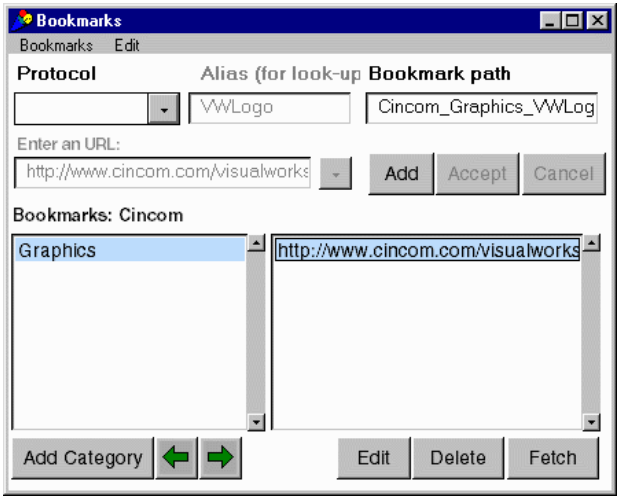
6 Enter an **Alias**. This is used to construct the Bookmark Path, which is what you will use in your applications. It can be any string. For this example, enter **VWLogo**.

7 Click the **Add** button.



The Bookmark Manager creates the bookmark and lists its URL in the URL list on the bottom right. VisualWave also generates a Bookmark Path for the new bookmark, although it does not automatically display it.

8 Click on the URL in the URL list to display the bookmark path in the **Bookmark path** field.



This is what you enter in your application to access the selected URL. In this case, the entire bookmark path is **Cincom_Graphics_VWogo**.

9 Save the new bookmark by choosing **Bookmarks→Save**. This makes the bookmark available for use in an application.

When prompted for a name, use the name that you have already given the bookmark set, Cincom. If you change the name, the first segment of each bookmark path in that set is changed to match the new set name.

You now have a set of bookmarks with one category and one bookmark.

Using Bookmarks

You can use bookmarks anywhere in an application that you would normally use an URL, such as for:

- The **Message** property for labels, radio buttons, check boxes, and action buttons.
- The **Background Pattern** and **Background Sound** properties for windows.
- The URL for an anchor within an HTML Text resource.
- The URL for a label that is being used as a link. The URL is entered on the **Web** properties page for the label.

In this next example, you will use bookmarks for the graphic image for a label and for a hyperlink for the label.

1 Open a new canvas for editing.

2 On that canvas, place a label widget.

3 Enter and apply the following properties for the widget:

Message	Cincom_Graphics_VWLogo
Label is Image	Checked

This assigns the graphic at the bookmarked URL as the label image.

4 Install the canvas on class **BookmarkTest** and selector **windowSpec**. When prompted to create the class, enter:

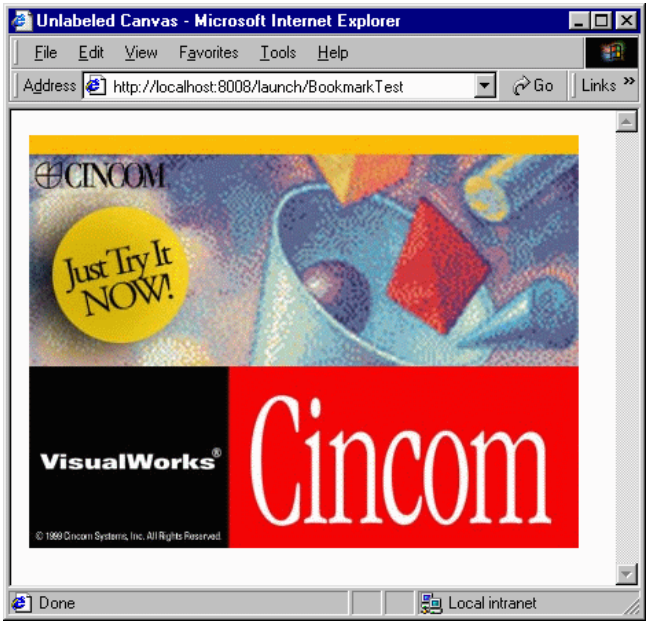
Name	BookmarkTest
Category	BookmarkDemos
Superclass	ApplicationModel

5 Test the application.

Start a TinyHTTPServer as a local host on port 8008. Make sure you have an open internet connections. Then open a browser and enter the URL:

http://localhost:8008/launch/BookmarkTest

The browser should display the Cincom graphic and logo.



6 To make the graphic a hyperlink to the Cincom web page, return to the canvas and enter and apply the following **Web** properties for the label widget:

Anchor	HREF
Protocol	HTTP
Enter an URL for the link	Default_VisualWave_Wave

You can load the Default bookmark set into the Bookmark Manager to view this definition.

Note that, with the Bookmark Manager open, you can copy a bookmark path into the Properties Tool using either of these shortcuts:

- Select the bookmark's URL in the URL list of the Bookmark Manager and choose **Copy Path** from the <Operate> menu. Then place the insertion point in the appropriate field of the Properties Tool and choose **paste** from the <Operate> menu.
- Select the bookmark's URL in the URL list of the Bookmark Manager and <Shift>-drag to the appropriate field of the Properties Tool.

7 Install the canvas on the class BookmarkTest and the selector widowSpec.

8 Test the updated label by as you did in step 5.

Notice that the graphic now has a border around it, and the cursor changes as you move it over the graphic. Click on the graphic to go to the Cincom web page.

Example: Updating Bookmarks

Using bookmarks in your application makes it easy to update the URLs to which those bookmarks refer. Returning to the example above, suppose that you wanted to display another graphic image, such as just the Cincom logo. To change the graphic image:

1 Open the Bookmark Manager.

2 Load the Cincom bookmark set.

3 Select the Graphics category and the URL to the graphic.

4 Click the Edit button. This enables the **Enter an URL** and **Alias** fields so that you can edit them.

5 In the **Enter an URL** field, enter the new URL to which this alias should refer. For this example, change the URL to:

http://www.cincom.com/images/topheader.gif

6 Click the **Accept** button and select **Bookmarks→Save** to update the bookmark set.

7 Run the application.

Notice that you did not have to change your application code; you only had to change the bookmark definition.

Troubleshooting

Unhandled exception: Bookmark: *name* not found

Make sure that the bookmark name is correct and that the bookmarks have been saved. When verifying the bookmark name, make sure that all the parts—bookmark set, category, and alias—are correct.

12

HTTP Cookies

HTTP Cookies are a general mechanism that web applications can use to both store and retrieve information on the client (web browser) side of the connection.

The following discussion presupposes that you are familiar with HTTP cookies. Information about HTTP cookies can be found in the following web sites:

- *Persistent Client State—HTTP Cookies* (http://www.netscape.com/newsref/std/cookie_spec.html) is the official specification from Netscape.
- *Cookies (Client-side Persistent Information) and Their Use* (http://home.netscape.com/assist/support/server/tn/cross-platform/20019.html) contains technical tips from Netscape.
- *SurfCart 1.1 Theory* and *How can I keep "state" information between calls to my CGI program?* (http://www.surfutah.com/surfcart/theory.html) (http://www.put.poznan.pl/hypertext/Internet/faq/www/hfields.htm) contain information about how HTTP cookies compare to other methods of storing state information and verifying clients.
- *SurfCart 1.1 Theory* and *How to make cookies and shopping cart* (http://www.surfutah.com/surfcart/theory.html) (http://users.ids.net/~oops/tech/make-cookie.html) describe how to use HTTP cookies to create a "shopping cart" for web catalog users.

- For some thoughts on security, see:
  Netscape tricks raise security concerns
  (http://www.macweek.com/mw_1011/gw_net_tricks.html).

Using HTTP Cookies

Each HTTP cookie is essentially a **`name=value`** pair that may contain additional information. This information is sent to the browser in the form of an HTTP header message and returned in the same fashion as an entity header.

VisualWave uses cookies to store the state of each active *session* (specifically, the session key and page key). For each web request, VisualWave keeps a dictionary of cookies.

To create your own cookies:

1. Create the cookie:

   ```
   HTTPCookie named: 'CookieName' value: aValue
   ```

   And fill in the required attributes.

2. Associate the cookie with a page:

   ```
   aWebPage addCookie: cookie
   ```

   This may appear in the application as early as its postBuildWith: method.

VisualWave automatically generates an HTTP metaheader that includes the cookie.

Setting Up the Web Page for Cookies

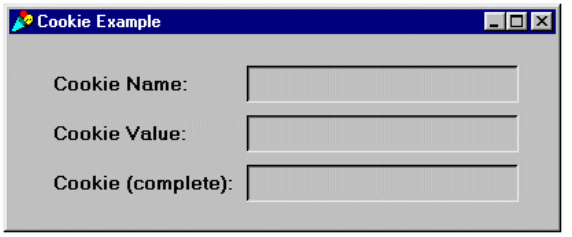
Cookies are attached to web pages or canvases by VisualWave. To begin working with cookies, let's create a simple canvas that displays the name of a cookie, its value, and its complete set of attributes.

1. Open a new canvas (by choosing **Tools → New Canvas** from the VisualWorks main window).
2. Place three labels on the left side of the canvas and three input fields on the right side of the canvas.
3. Assign the following properties to the widgets:

Widget	Property	Value
Window	**Label**	Cookie Example

Label	**Label**	Cookie Name:
Label	**Label**	Cookie Value:
Label	**Label**	Cookie (complete):
Input Field	**Aspect**	#cookieName
	ID	#cookieNameField
	Details	Bordered, Can Tab, Read Only
Input Field	**Aspect**	#cookieValue
	ID	#cookieValueField
	Details	Bordered, Can Tab, Read Only
Input Field	**Aspect**	#cookieComplete
	ID	#cookieCompleteField
	Details	Bordered, Can Tab, Read Only

4 Resize the canvas and set its size (**Layout→Window→Preferred Size**). Your canvas should now look like this:



5 Install the canvas on class CookieExample and selector windowSpec. When prompted to create the CookieExample class, enter the following characteristics:

Name:	CookieExample
Category:	Examples
Superclass:	ApplicationModel

6 Select all the input fields and click on the **Define...** button of the Canvas Tool. Define models and initialization for cookieName, cookieValue, and cookieComplete.

At this point you have the class CookieExample. That class has one class method that returns the windowSpec. It has an instance variable and an accessor method for each of the three aspects cookieName, cookieValue, and cookieComplete.

Adding a Cookie to the Page

To add a cookie to a page, you simply create the cookie and then attach it to the web page. For this example, you'll create a cookie that displays the number of times that the user has visited this web page.

1. Open a System Browser and display the CookieExample class.
2. Add a protocol (method category) called interface opening.
3. Enter and accept this new method:

```
postBuildWith: aBuilder

    | cookie |
    cookie := HTTPCookie named: 'VisitsToCookieExample'
    value: '1'.
    cookie expireAfterDays: 1.
    cookie path: '/launch/CookieExample'.

    aBuilder window addCookie: cookie.

    self cookieName value: cookie name.
    self cookieValue value: cookie value.
    self cookieComplete value: cookie valueString
```

Within this method:

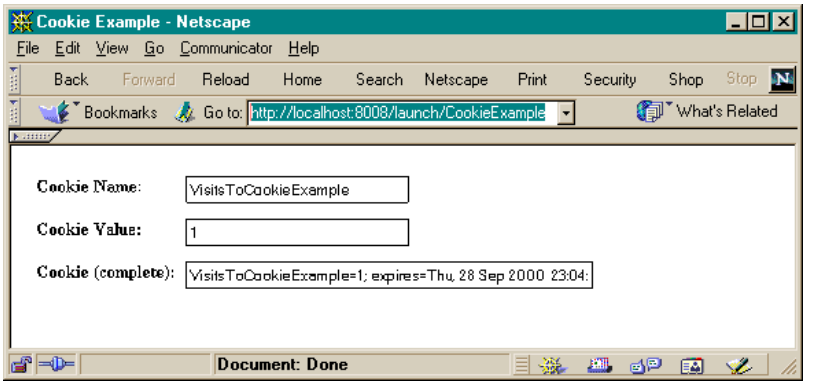
HTTPCookie named:value:
Creates a new HTTP cookie with the given name and value. The name must be a String; the value can be anything. The methods expiresAfterDays: and path: set additional attributes of the cookie.

aBuilder window addCookie: cookie
Attaches the cookie to the web page so that VisualWave will generate the appropriate HTTP header for the cookie when it generates the HTML for the web page.

self cookieName value: cookie name,
self cookieValue value: cookie value, and
self cookieComplete value: cookie valueString

Update the cookieName, cookieValue, and cookieComplete fields to display the new cookie's name, value, and complete set of attributes. Note that cookies are not normally displayed on the screen to the user. Here you explicitly updated the display to show the cookie that you created.

4 Run the CookieExample from a web browser. Notice that the values you gave to the cookie are correctly displayed. Notice the difference between the cookie's value and its valueString.



Displaying All the Cookies Being Sent

In the previous example, you learned how to create a cookie and how to display your cookie's name and value in a web page.

There are, however, other cookies attached to your web page. VisualWave attaches two cookies to every web page: sessionKey and pageKey. VisualWave uses those cookies to determine where the user left off and to return that page if the user enters a generic **`submit`** (http://hostname:port/submit) to VisualWave.

You can get the complete set of cookies that is being sent with the web page by sending the message cookies to the window's HTMLEntity:

```
self builder window htmlEntity cookies
```

Obviously, you need to send this message *after* VisualWave has built the web page and included all possible cookies, including the sessionKey and pageKey. You can do this by implementing a noticeOfAnswer: method on your application model.

1 Open the CookieExample windowSpec for editing.

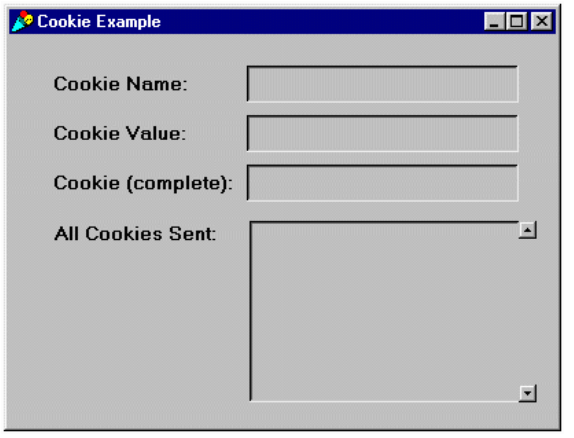
2 Resize the canvas and then set its new size using **Layout→Window→Preferred Size**.

3 Add a label on the left side of the canvas and a text editor field on the right side of the canvas.

4 Assign the following properties to the widgets:

Widget	Property	Value
Label	**Label**	All Cookies Sent:
Text Editor	**Aspect**	#cookiesSent
	ID	#cookiesSentField
	Details	Bordered, Can Tab, Read Only

Your canvas should now look like this:



5 Install the canvas in class CookieExample and selector windowSpec.

6 Select the Text Editor widget next to **All Cookies Sent:** and press the **Define...** button. Define the model for cookiesSent, including initialization.

7 In a System Browser, display the CookieExample class.

8 In the interface opening protocol, enter and accept this method:

```
noticeOfAnswer: aWebPage

        | tmpStrm |

        tmpStrm := ReadWriteStream on: String new.
        aWebPage htmlEntity cookies do:
            [ :each | tmpStrm nextPutAll: each valueString; cr ].
        cookiesSent value: tmpStrm contents.

        ^aWebPage
```

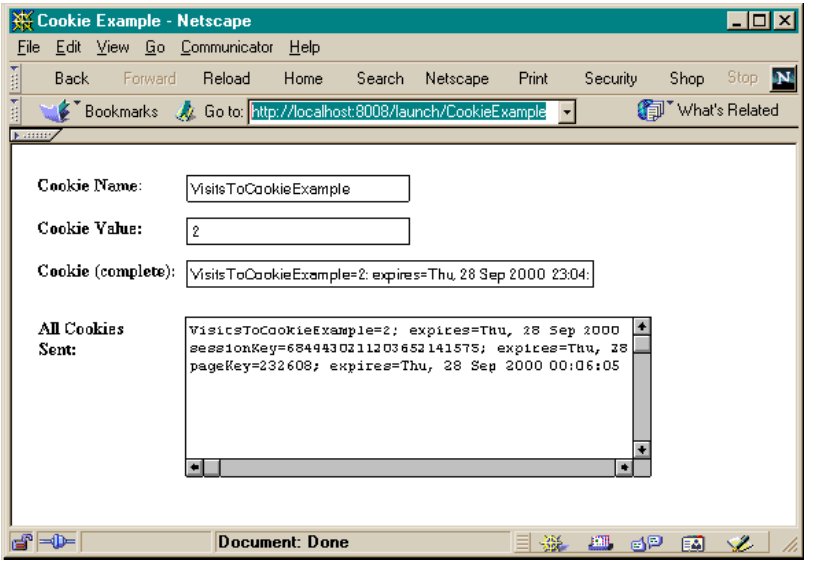
Within this method:

aWebPage htmlEntity cookies
returns the complete set of cookies that are associated with the current web page.

tmpStrm holds the string representation of the set of cookies.

cookiesSent value: tmpStrm contents
updates the **All Cookies Sent** field to display the complete set of attributes for every cookie. Note that cookies are not normally displayed on the screen to the user. Here you explicitly updated the display to show the cookie that you created.

9 Run the application.



Accessing the Cookies in the Request

You've learned how to create and add cookies to a web page and how to verify the set of cookies that are being sent with the web page. Your application can also access the content of the cookies stored by the client's web browser. Whenever the web browser makes a request, it checks the domain and path to see if they match any associated with the cookies that it is holding. If there is a match, it sends the appropriate cookie(s) with the request. Those cookies are part of the web request in VisualWave. An application model can get the cookies from the incoming web request using this message-send:

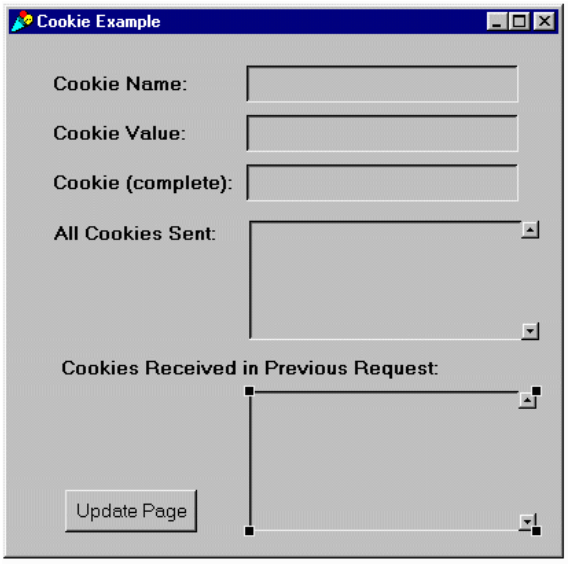
```
self webRequest cookieData
```

To see the results, let's expand the CookieExample to include a field for the cookies that were in the previous web request.

1. Open the CookieExample windowSpec for editing.
2. Resize the canvas and then set its new size using **Layout→Window→Preferred Size**.
3. Add a label on the left side of the canvas and a text editor field on the right side of the canvas. Add an action button at the bottom left corner.
4. Assign the following properties to the widgets:

Widget	Property	Value
Label	**Label**	Cookies Received in Previous Request:
Text Editor	**Aspect**	#cookiesReceived
	ID	#cookiesReceivedField
	Details	Bordered, Can Tab, Read Only
Action Button	**Label**	Update Page
	Action	#submit
	ID	#submitButton

Your canvas should now look like this:



5 Install the canvas in class CookieExample and selector windowSpec.

6 Define cookiesReceived, including initialization. Do not define submit.

7 In a System Browser, display the CookieExample class.

8 In a new actions protocol, enter and accept this method:

```
submit
    ^self
```

9 Edit the noticeOfAnswer: method so that it contains this code:

```
noticeOfAnswer: aWebPage

        | tmpStrm tmpStrm2 aReq |

        tmpStrm := ReadWriteStream on: String new.
        aWebPage htmlEntity cookies do:
            [ :each | tmpStrm nextPutAll: each valueString; cr ].
        cookiesSent value: tmpStrm contents.

        tmpStrm2 := ReadWriteStream on: String new.
        aReq := self webRequest.
        aReq cookieData notNil ifTrue:
```

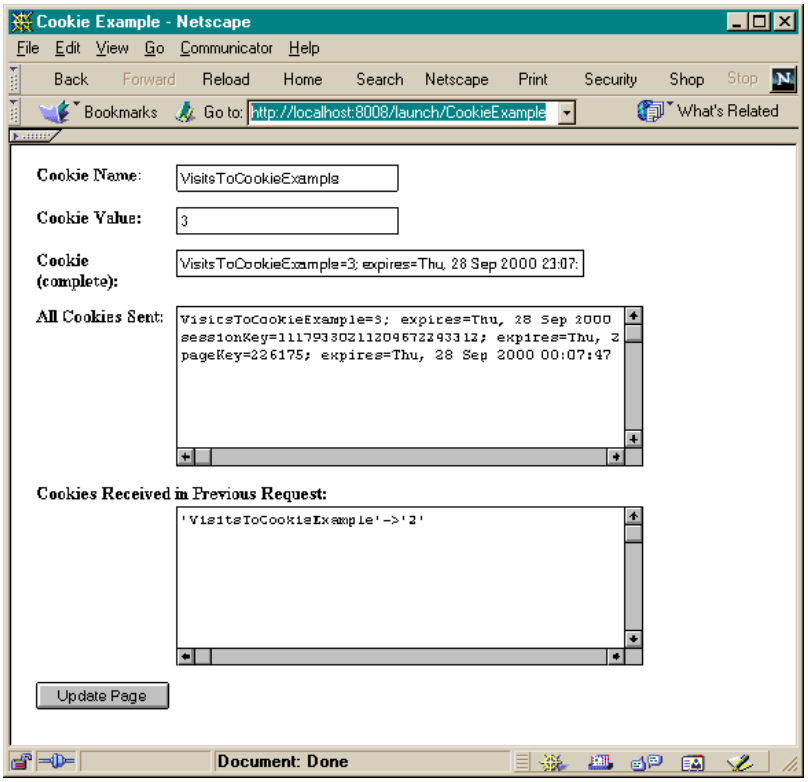
```
        [aReq cookieData orderedNameValues
            do: [ :e | tmpStrm2 nextPutAll: e printString; cr ]].
    cookiesReceived value: tmpStrm2 contents.

    ^aWebPage
```

10 Start the application.

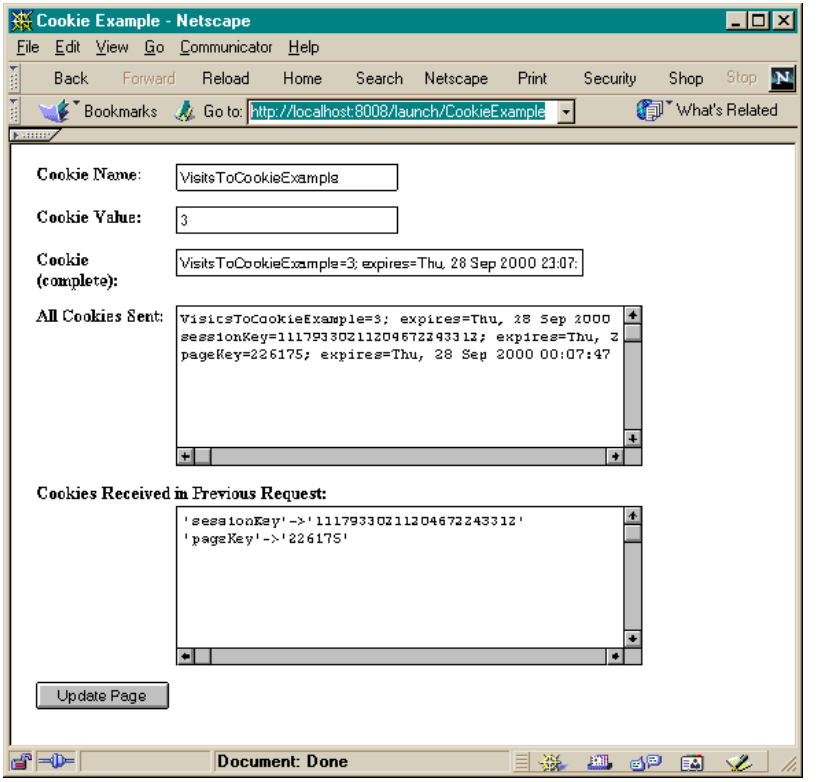
When you launch the application, the page shows that three cookies were sent with this page and that one cookie was sent with the launch request (see image below). That cookie is the VisitsToCookieExample cookie that you set the last time you ran the application from the web browser. This time when you ran the application, the browser checked its cookies and found one that matched the path **`/launch/CookieExample`** so it sent it with the launch request.

Notice that the sessionKey and pageKey cookies that were sent with this request have a path of **`/submit`**. That means that if the browser does a submit, it should include these cookies along with the transaction.



11 Click the **Update Page** button. The page returned by VisualWave now has the sessionKey and pageKey cookies in the list of cookies that were sent with the previous request. The VisitsToCookieExample cookie is

not listed because it is only sent to VisualWave in a launch request; this was a submit request.



Using Received Cookies

As the application is now, the VisitsToCookieExample is always 1. Ideally, the application should check for a cookie sent by the browser in the launch request. If the cookie is sent, the application should add one to the cookie and send it back to the browser. If not, it should send the cookie. To make that happen, change the method postBuildWith: as follows:

```
postBuildWith: aBuilder

    | cookie oldCookie aReq |
    aReq := self webRequest.

    oldCookie := aReq cookieValueAt: 'VisitsToCookieExample'.
    oldCookie isNil
        ifTrue: [cookie := HTTPCookie
            named: 'VisitsToCookieExample' value: 1]
        ifFalse: [cookie := HTTPCookie
            named: 'VisitsToCookieExample' value:
                (oldCookie asNumber + 1)].
    cookie expireAfterDays: 1.
    cookie path: '/launch/CookieExample'.

    aBuilder window addCookie: cookie.

    self cookieName value: cookie name.
    self cookieValue value: cookie value.
    self cookieComplete value: cookie valueString.
```

Class HTTPCookie

Class HTTPCookie provides a general mechanism that VisualWave applications can use to both store and retrieve information on the client (web browser) side of the connection. Each HTTPCookie is essentially a name=value pair that may contain additional information. This information is sent to the browser in the form of an HTTP header message and returned in the same fashion as an entity header.

An instance represents an HTTP cookie.

Instance variable	HTML attribute	Description
name (String)	name=	Name of the cookie
value (String)	value	Value of the cookie
expires (Timestamp)	expires=*date*	When this time is reached, the browser may discard the cookie [optional]
domain (String)	domain=*domain Name*	The browser must be speaking to this domain before it will return the cookie [optional]

path (String)	path=*pathName*	This must be a prefix of the URL path before the browser will return the cookie [optional]
secure (Boolean)	secure	If this is true, the cookie will only be returned if we are using a secure server

During initialization for a new HTTPCookie instance, secure is set to false.

The expires TimeStamp should be in local time.

Examples

CookieDemo is much like the application described in this chapter, with the addition of account number and page count features. CookieDemo is in the VisualWaveDemos parcel.

13

Client Pull

This section describes VisualWave's support for client pull technology.

You and your application's users must have a web browser that supports client pull.

This section assumes that you are familiar with the client pull feature of the HTTP protocol, the HTML format, and web browsers. Information about client pull can be found in numerous web sites. A few are listed here:

- *An Exploration of Dynamic Documents* (http://www.netscape.com/assist/net_sites/pushpull.html) is the official story from Netscape.

Overview of Client Pull

Normally, web browsers are driven by user input. A user clicks on a link or requests an URL and the HTTP server returns the appropriate data. Each exchange is inititated by the user.

In non-web software applications, the application may respond to requests from the user (just as with a web browser), but the application may also initiate interaction with the user. Applications commonly interrupt the user to display error messages or to update the display to reflect new data as it becomes available.

Obviously, it would be nice for the application to have that form of communication with users who access it from a web browser. Client pull is one way your application can take independent action.

With client pull, the server transmits page information to the user's web browser that automatically instructs the web browser to perform an action such as "reload this page in ten minutes" or "go load this URL in two

minutes." After the specified amount of time has elapsed, the web browser pulls updated pages according to the instructions provided along with the page. The user can terminate the page's actions by closing the page.

Client pull differs from server push in that:

Server push
: The server sends data to the web browser. The browser displays the data and leaves the connection open. The server uses that open connection to send more data. Whenever the server sends more data, the browser displays it, still leaving the connection open. The connection remains open and controlled by the server until either the server or the user breaks the connection.

Client pull
: The server sends data to the web browser. That data includes a directive (in the HTTP response or the document header) that instructs the browser to reload the current data or load new data after a specified amount of time. After the initial data is sent, the connection is closed. After the specified amount of time has elapsed, the browser follows the instructions, opening a new connection and either reloading the current data or getting new data.

Client pull is most useful when you want to provide the user with up-to-date information as soon as that information becomes available. A simple use of client pull is to cause a document to be automatically reloaded on a regular basis.

For example, client pull can be used to implement:

- A weather watcher that shows an updated satellite photo at 15-minute intervals.
- A stock ticker that displays a new quote data every 5 minutes.

There are hundreds of other examples on the World Wide Web. Whenever images start to flash in your browser and you feel that you've lost control, you've probably loaded a page that makes use of server push or client pull. If you've set your browser to display the current URL and that URL changes as the image changes, you're probably observing client pull in action.

Using Client Pull

Basic HTTP Refresh

1. Open the canvas for the window that you want to refresh.
2. Display the canvas's properties.
3. Select the **Web** properties page.
4. In the **Refresh every x seconds** field, enter the amount of time between refreshes. The value must be equal to or greater than one.
5. Apply the properties.
6. Install the canvas.
7. Run the application from a web browser that supports client pull done with HTML `<Meta>` headers and HTTP refresh.

Example: Letting the User Set the Refresh Rate

To demonstrate the features of client pull, let's create a simple application that enables the user start client pull, stop client pull, and set the refresh rate. The application will display the time the page was sent and the number of times that the page has been refreshed.

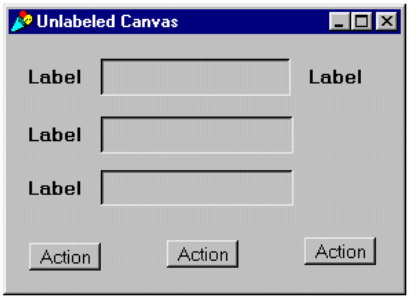
This example is divided into several high-level tasks:

1. Create the canvas.
2. Set up the time, count, and refresh rate displays.
3. Set up the screen and web variations.
4. Start client pull.
5. Stop client pull.
6. Run the application.

Create a Simple Canvas

1 Open a new blank canvas for editing.

2 On the canvas, place four labels, three input fields, and three action buttons, as shown here:



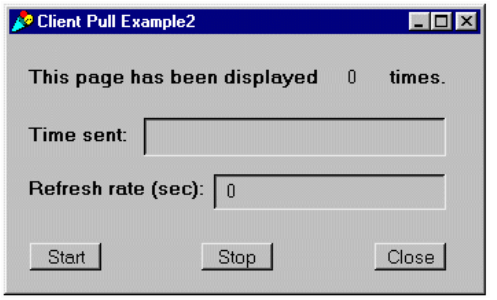
Use the Properties tool to set the properties for the window and widgets as shown in this table. Where a property is not specifically mentioned, use the default value.

Component	Properties Page	Properties
Window	**Basics**	Label: ClientPullExample2
	Web	Refresh every x seconds(*)
Label (upper left)	**Basics**	Label: This page has been displayed
Label (upper right)	**Basics**	Label: times.
Label (middle left)	**Basics**	Label: Time sent:
Label (lower left)	**Basics**	Label: Refresh rate (sec):
Input field (top)	**Basics**	Aspect: #count ID: #countField Type: Number
	Details	Bordered: not checked
Input field (middle)	**Basics**	Aspect: #timestamp ID: #timeField
Input field (bottom)	**Basics**	Aspect: #refreshRate ID: #refreshRateField Type: Number
Action button (left)	**Basics**	Label: Start Action: #startRefresh ID: #startButton

Action button (middle)	**Basics**	Label: Stop Action: #stopRefresh ID: #stopButton
Action button (right)	**Basics**	Label: Close Action: #closeRequest

(*) This property specifies the initial client pull refresh rate for the window. Because you don't want the window to refresh until the user starts client pull, leave this value 0.

3 Resize and arrange the widgets as desired.



4 Set the window's default size by choosing **Layout→Window→Preferred Size**.

5 Install the canvas by clicking the **Install** button on the Canvas Tool. Install the canvas in class ClientPullExample2 and selector windowSpec. When prompted, choose to create class ClientPullExample2 in category ClientPullDemos as an Application and subclass of ApplicationModel.

6 Ensure that nothing is selected in the canvas. Define the canvas by clicking the **Define** button on the Canvas Tool. Choose to define the model and add initialization for count, refreshRate, startRefresh, stopRefresh, and timestamp. Do not define closeRequest.

At this point, you have set up the interface for the example. In the next sections, you will learn how to program the widgets to actually display the current time and count, start client pull, and stop client pull.

Set up the Time Display

1 In a System Browser, display the ClientPullExample2 method timestamp (in the protocol aspects).

2 Edit the method to be the following:

```
timestamp
  ^timestamp isNil
    ifTrue: [timestamp := Time now printString asValue]
    ifFalse: [timestamp]
```

3 Accept the change.

Set up the Count Display

1 In a System Browser, display the ClientPullExample2 method count (in the protocol aspects).

2 Edit the method to be the following:

```
count
  ^count isNil
    ifTrue: [count := 1 asValue]
    ifFalse: [count]
```

3 Accept the change.

Set up the Refresh Rate Display

1 In a System Browser, display the ClientPullExample2 method refreshRate (in the protocol aspects).

Notice that the default value for the refreshRate input field is 0 seconds. Note that the initial value is *not* the refresh rate when the window is first opened; it is a suggestion for the user's input. The initial refresh rate is stored in the window property **Refresh every x seconds**.

2 Change the refreshRate method so that the initial value displayed is the initial refresh rate from the window's properties by editing the refreshRate method so that is matches the method shown here:

```
refreshRate
    ^refreshRate isNil
        ifTrue: [refreshRate :=
            (self builder windowSpec propertyAt: #refreshSeconds)
            asValue]
        ifFalse: [refreshRate]
```

3 Accept the change.

Set up Screen and Web Variations

Now let's set up the code that starts and stops client pull so that it works when the application is run from the screen as well as from a web browser. The dual behavior is made possible by a watcherProcess in combination with refreshForScreen and refreshForWeb methods.

1. Display the class definition for ClientPullExample2. Add the variable watcherProcess to the list of instanceVariableNames (which already includes count, timestamp, and refreshRate) and accept the change.
2. Create a new protocol called accessing and add the following instance method:

```
watcherProcess
    ^watcherProcess
```

3. Create a new protocol called submitting and add the following instance method:

```
refreshForScreen
    self count value: count value + 1.
    self timestamp value: Time now printString.
    self builder window repairDamages
```

4. In the protocol called submitting, add the following instance method:

```
refreshForWeb
    self count value: count value + 1.
    self timestamp value: Time now printString
```

5. In the protocol called submitting, add the following instance method:

```
submitFrom: submitController toComponents:
    componentCollection
"If this submit is a refresh event from the browser, do the
refresh."

 submitController == self builder window controller
    ifTrue: [self refreshForWeb].
^super submitFrom: submitController
    toComponents: componentCollection
```

Start Client Pull

1 In the System Browser, display the startRefresh method in ClientPullExample2 (in the protocol actions).

2 Edit the method so that it looks like this:

```
startRefresh
    ProcessEnvironment
        executeForScreen:
            [self refreshForScreen.
                watcherProcess := [
                [(Delay forSeconds: self refreshRate value) wait.
                    self timestamp dependents isEmpty]
                        whileFalse: [self refreshForScreen]]
                        forkAt: Processor userBackgroundPriority].
ProcessEnvironment executeForWeb:
    [self builder window refreshAfterSeconds:
        self refreshRate value]
```

3 Accept the change.

If the application is run from a web browser, the start method tells the web page (self builder window) to set refreshAfterSeconds to be the refreshRate that the user entered. If the refreshRate is at least 1, a web page is generated and returned after the user's submit. This page contains the following: an HTML header with **`HTTP-Equiv=Refresh`**; the content set to refreshRate; and the URL needed to redisplay the web page.

If the application is run from the screen, it waits the amount of time designated in the refreshRate, then calculates a new timestamp value, and then redisplays the window.

When the user submits the page, the WebSession sends submitFrom:submitController toComponents: componentCollection to the class. The submitFrom:toComponents: method checks to see if the submitController is the same as the WebPage's controller. If it is, then it refreshes the timestamp. If it isn't, then it does not refresh the timestamp. In either case, it then does a

```
^super submitFrom: toComponents:
```

which causes it to return the componentCollection.

Stop Client Pull

1 In the System Browser, display the stopRefresh method in ClientPullExample2 (in the protocol actions).

2 Edit the method so that it looks like this:

```
stopRefresh
    ProcessEnvironment
        executeForScreen:
            [self watcherProcess terminate.
            watcherProcess := nil].
    ProcessEnvironment executeForWeb:
        [self builder window noRefresh]
```

3 Accept the change.

Running the Example

1 In VisualWave, use the Server Console to create and start a TinyHttpServer.

2 In a web browser, request an URL such as the following, substituting the host and portname of your TinyHttpServer for localhost:8008 if necessary:

```
http://localhost:8008/launch/ClientPullExample2
```

3 Enter a refresh rate that is fairly small, such as 5.

4 Click the **Start** button.

The page is submitted and another is returned. Wait until the refresh interval and watch the page automatically submit itself.

5 Click the **Stop** button.

Additional Examples

For a description of the client pull examples in this chapter, see “Using Client Pull” on page 117, and study the other examples included in the VisualWaveDemos parcel.

14

FileResponder Resolver

VisualWave includes a resolver called a *FileResponder*. The FileResponder resolver appears in the Server Console on the **Resolver Type** menu when you are editing or creating a resolver.

A FileResponder is a general mechanism to enable the server to serve out files referenced in VisualWave applications. One reason to use this is if the server has the responsibility of serving out embedded content which is housed in files.

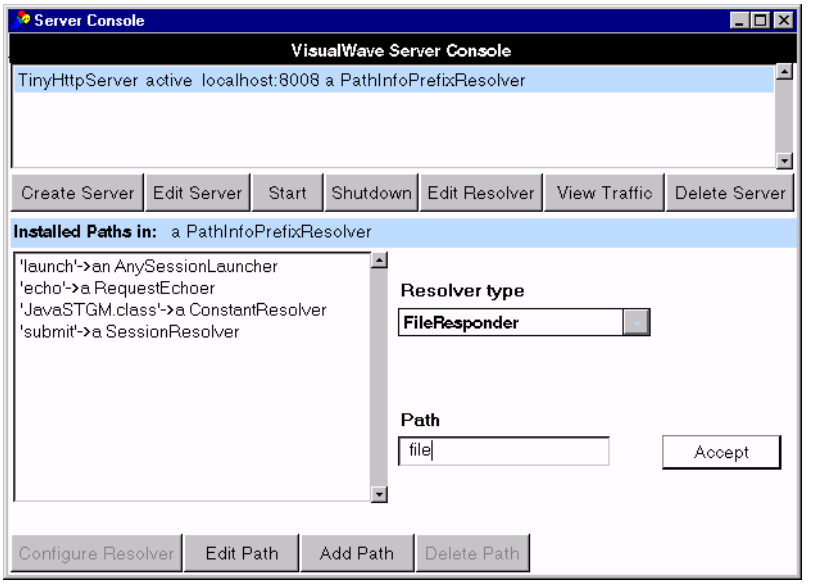
An example of an external file that can be retrieved using a FileResponder is a VRML world (`.wrl`). In VRML, there may be many different kinds of embedded content. VRML worlds may have subworlds specified as "`WWWInlines`," Node textures as "`Material`," and others. To a VRML-enabled browser this means inflating this inline or material in the same scene that is being constructed. These imbedded pieces are in turn requested from the server using file references. The FileResponder is responsible for handling these file requests and bundling up file content to send back to the client browser for construction to continue.

Configuring a FileResponder

To use a FileResponder:

1. In the Server Console, select a server.
2. Display the server's resolvers by clicking the **Edit Resolver** button.
3. To add a new resolver, click the **Add Path** button.
4. To make the new resolver a FileResponder, pull down the **Resolver type** menu and choose **FileResponder**.

5 In the **Path** input field, enter the string that will be included in URLs to cause VisualWave to serve files. In this example, the path is **file**.



6 Accept the new resolver.

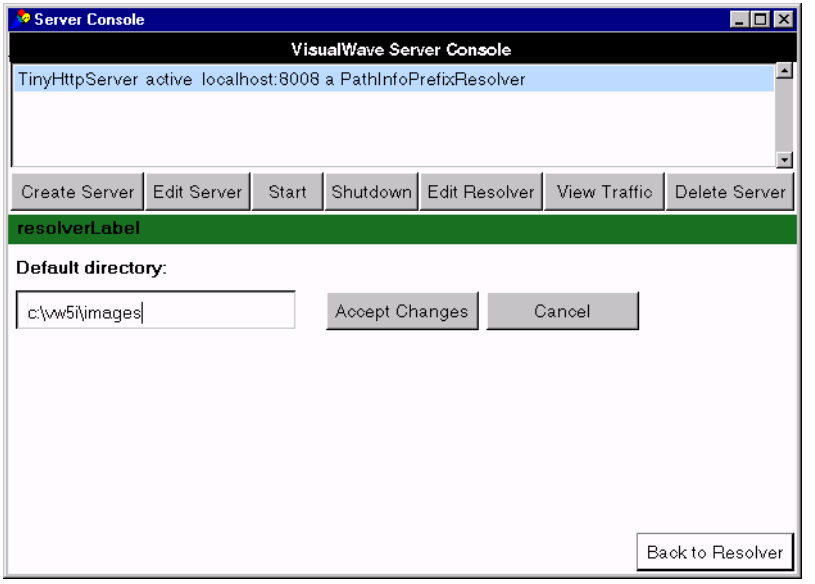
Now, whenever the server gets a request which has the path prefix file, it will use the FileResponder to service the request.

To configure the FileResponder:

1 Select its entry in the list of **Installed Paths**.

2 Click the **Configure Resolver** button.

3 In the **Default directory** input field, enter the name of the directory in which the FileResponder is to look for files.



4 Accept the change.

Content Types

The FileResponder creates a content type based on the file extension in the requested path. The current types supported are:

Category	file extension	content type
Text types:	`.htm`	text/html
	`.html`	text/html
	`.txt`	text/plain
	`.text`	text/plain
Application types:	`.js`	application/x-javascript
	`.mocha`	application/x-javascript

VRML	**.wrl**	x-world/x-vrml
Video	**.avi**	video/x-msvideo
	.mov	video/quicktime
	.qt	video/quicktime
	.moov	video/quicktime
	.mpeg	video/mpeg
	.mpg	video/mpeg
	.mpe	video/mpeg
	.mpv	video/mpeg
	.vbs	video/mpeg
	.mpegv	video/mpeg
Audio	**.wav**	audio/x-wav
	.au	audio/basic
	.snd	audio/basic
	.mp2	audio/x-mpeg
	.mpa	audio/x-mpeg
	.abs	audio/x-mpeg
	.mpega	audio/x-mpeg
	.aif	audio/x-aiff
	.aiff	audio/x-aiff
	.aifc	audio/x-aiff

Images	**.bmp**	image/x-MS-bmp
	.rgb	image/x-rgb
	.gif	image/gif
	.ief	image/ief
	.png	image/x-png
	.pcd	image/x-photo-cd
	.ppm	image/x-portable-pixmap
	.pgm	image/x-portable-graymap
	.pbm	image/x-portable-bitmap
	.pnm	image/x-portable-anymap
	.xwd	image/x-xwindowdump
	.xpm	image/x-xpixmap
	.xbm	image/x-xbitmap
	.ras	image/x-cmu-raster
	.tif	image/tiff
	.tiff	image/tiff
	.jpeg	image/jpeg
	.jpg	image/jpeg
	.jpe	image/jpeg
	.pip	image/jpeg
	.jfif	image/jpeg
	.pipeg	image/jpeg

You can add to these types by sending a message to the FileResponder class:

```
FileResponder addType: 'application/octet-stream'
        extension: #exe.
```

or modify the FileResponder>>initializeDefaultTypes method to always include your new type.

15

Graphic Image Formats

VisualWave incorporates the GIF-compatible GUF format and a Java Rendered graphics format. This section describes the three options available for using graphic images:

- Automatically Generating Graphic Images
- Using the Java Renderer
- Using GIF-compatible (GUF) Images

> **Note:** Due to licensing restrictions, the standard VisualWave image does not include the ability to generate true compressed GIF images. For information about how to obtain the GIF compression code, see the VisualWave Release Notes.

To use the Java Renderer, you and your application's users must have a web browser that supports Java and Java must be enabled in the browser.

To use the GIF-compatible GUF format, you and your application's users must have a web browser that supports standard GIF images. Almost all web browsers meet this requirement.

Automatically Generating Graphic Images

VisualWave automatically generates graphic images in several situations:

- Smalltalk graphic images (instances of any subclass of Image) are automatically converted to a web-compatible graphics format.
- Some widgets for which there is no HTML equivalent are automatically converted to web-compatible graphic images and incorporated using the HTML entity closest to the widget's original purpose. In particular, the following widgets are converted to web-compatible graphic images:
  - Buttons for which the **Enforce Boundary with GIF** property is set.
  - Menu bars
  - Charts
  - View holders
- Widgets for which the label or background is a graphic image can use either an external graphic image (referenced by URL) or a Smalltalk graphic image. When they use a Smalltalk graphic image, that image is converted to a web-compatible graphics format.

If you have a static graphic image that is used in your application, it is more efficient to place that image in an external file and reference it by URL or bookmark. Doing so speeds up access in two ways:

- VisualWave does not need to generate the graphic image and send it to the web browser each time it is used. Instead, VisualWave sends an URL and the web browser retrieves the image.
- The web browser caches the image for reuse. An image generated by VisualWave may not have the same URL each time that it is generated, causing the web browser to reload the image each time it is used.

Using the Java Renderer

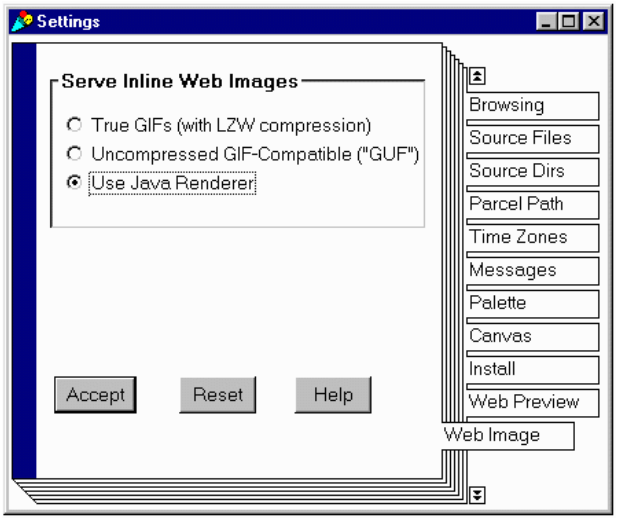
VisualWave can generate graphic images using a Java Renderer. When you use the Java Renderer, a Java applet is downloaded to the user's web browser. VisualWave then sends instructions to that applet, describing the graphic image that is to be rendered. Java rendered graphic images download and display more quickly than VisualWave's GIF-compatible GUF images. Java Rendered graphic images, however, require that the user's web browser support Java and that Java is enabled in that browser. Furthermore, they may not be as crisp as GIF-compatible images.

By default, VisualWave generates graphic images in GIF-compatible GUF format. To cause VisualWave to generate graphic images using the Java Renderer:

1 Display the Settings Tool (in the main window choose **File→Settings…**).

2 Display the **Web Image** page.

3 Set the **Use Java Renderer** option.



4 Apply the settings.

5 Save the VisualWave image.

Once **Use Java Renderer** is set, all graphic images generated by VisualWave are sent to the user's browser with a Java Renderer.

Note: At this time, masked images are not supported by the Java libraries and cannot be rendered using the Java Renderer.

Using GIF-compatible (GUF) Images

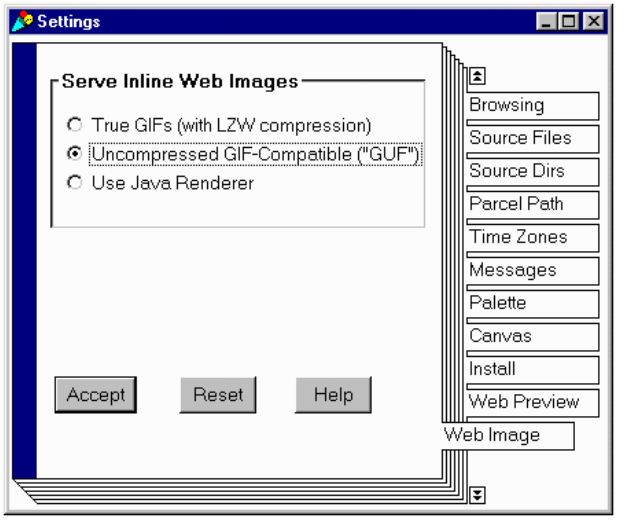
The GUF format is the same as *un*compressed GIF format. As a result, GUF images can be displayed in any web browser that can display GIF images. GUF images, however, are as much as 2 to 20 times as large as GUF images. As a result, they take more time to download and display than compressed GIF images.

By default, VisualWave generates graphic images in GIF-compatible GUF format. If you have set VisualWave to use the Java Renderer, you can reset it to use the GUF format:

1 Display the Settings Tool (in the main VisualWave window choose **File→Settings**.

2 Display the **Web Image** page.

3 Set the **Uncompressed GIF-Compatible (GUF)** option.



4 Apply the settings.

5 Save the VisualWave image.

16

ActiveX Widget

This section describes VisualWave's support for ActiveX. Developers and application users must have a web browser that supports ActiveX.

Browser Rrequirements:

ActiveX support is native in Microsoft Internet Explorer Version 3.0 and higher. Netscape Navigator may require a special plug-in available from the Netscape web site.

Using ActiveX

VisualWave supports inclusion of, but not communication with or between, ActiveX components. To add an ActiveX component to a canvas:

1. Open a canvas for editing.
2. From the Palette, choose the ActiveX widget  and place it on the canvas. Inactive Java/ActiveX regions appear onscreen as labelled gray areas.
3. Set the properties for the widget (see table below).
4. Install the canvas.

ActiveX Properties

Page	Property	Required	Description
Basics	ID	No	The name of this component.
	Class ID	Yes	An URL whose syntax depends on the type of ActiveX's object being embedded. Registered ActiveX controls begin with `CLSID:class identifier`.
	Code Base	Yes	An URL or series of URLs that identify the location of the ActiveX files. Thus, the code must be accessible from some web server.
	Data	No	An URL that identifies the data for the object.
	HTML ID	Yes	The HTML `ID` attribute for the ActiveX object.
	Alternate HTML	No	When checked, the HTML in the text field below is shown in web browsers that cannot run ActiveX components.

Parameters	Parameters	No	Enables you to pass information to the Java applet. Parameters should be entered one per line, in the format:
			`name=value`
			`name2=value2`
			`name3=value3`

Setting Properties Programmatically

In addition to being set from the Properties tool, ActiveX parameters can be set programmatically in the instance of HTMLInsertObject that represents the ActiveX widget. For example:

```
postOpenWith: aBuilder
    "Add a parameter named 'BackToWaveUrl' with a value
    that contains an url that represents this page with
    an additional jabber component."

    self executeForWebOnly:
        [(self builder componentAt: #wavebackRegion)
        widget htmlEntity add:
            (HTMLParameter new
                name: 'BackToWaveUrl';
                value: (self builder window urlForRefresh ,
                    '&jabber='))].
```

Only a minimal set of ActiveX `<OBJECT>` tag parameters are explicitly available in the Properties tool. The other parameters can be set programmatically in the instance of HTMLInsertObject that represents the ActiveX widget. For example, use the following code fragment to set the vspace parameter of the HTMLInsertObject:

```
postBuildWith: aBuilder
   | webJKGTable |
   webJKGTable := aBuilder componentAt: #table.
   webJKGTable htmlEntity vspace: 8
```

Notifying Browsers without ActiveX

You can specify the message to be displayed in browsers that do not support ActiveX components. To do so:

1. Open for editing the canvas that contains the ActiveX widget.
2. Select the ActiveX component.
3. Display the ActiveX component's properties.
4. In the Properties Tool, display the **Basics** page.
5. On the **Basics** page, check the **Alternate HTML for non-ActiveX browsers** checkbox.
6. In the text field below the checkbox, there is a default message. You can edit the message to suit your situation. The message can include any standard HTML markup.
7. Apply the changes.
8. Install the canvas.

Examples

VisualWave comes with two examples using embedded ActiveX controls:

- **ActiveXDemos** with the default window specification.
- **ActiveXDemos** with the textSpec window specification. To run this application, request an URL of the form:

  http://host:path/launch/ActiveXDemos?textSpec.

The ActiveX demos are in the VisualWaveDemos parcel.

A

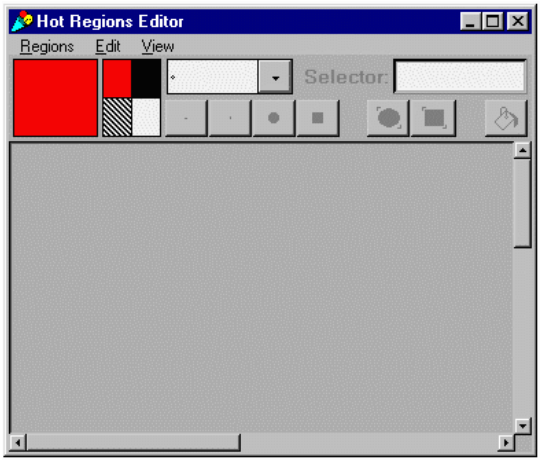
Tools Reference

This appendix contains a complete description of each of the tools in the VisualWave environment:

- Hot Regions Editor
- HTML Text Editor
- Bookmark Manager

Hot Regions Editor

Use the Hot Regions Editor to create and edit a hot region mapping, which can be integrated into a click map widget.



Menu Bar Commands

Regions Menu

New

Clears the Hot Regions Editor of the current set of region mappings and prepares for a new set.

Load

Loads the hot regions resource from the specified class and selector into the Hot Regions Editor.

Install

Prompts for the method selector and class name where hot regions specifications are stored.

Read

Edits hot regions that have been applied to a selected widget.

Apply

Applies the hot regions in the Hot Regions Editor to an associated canvas.

Exit

Quits the Hot Regions Editor.

Edit Menu

Undo Painting

Reverses the most recent painting operation.

New Slice

Ends the editing of the last hot region and enables editing of a new hot region.

Clear Slice

Clears all of the areas specified in the current region. The region's name and selector remain.

Delete Slice

Deletes the current region. Deletes the region name, selector, and areas specified.

View Menu

Load Backdrop…

Prompts for the background image to display. This image can be used as a guide when drawing regions, but is not connected to the hot regions. The association between an image and hot regions is made by specifying them both in the properties of a click map widget.

All Slices

Displays all of the slices for the current hot region resource at one time. Useful for looking for areas that do not have a hot region defined.

Painting Controls

You paint slices in the Hot Regions Editor much like you use other painting applications. A single slice may contain any number of areas, connected or not connected.

The Hot Regions Editor contains several painting controls.

Ink selector:

Allows you to choose the color or pattern of ink for painting. The color you choose does not show up in the running click map; it is only for your convenience when specifying regions.

Brush 1:

The smallest size paintbrush, allowing you to paint very small areas.

Brush 2:

The next smallest size paintbrush.

Brush 3:

A larger, circular paintbrush.

Brush 4:

A larger, square paintbrush.

Ellipse:

When chosen, enables you to draw filled elliptical areas.

Rectangle:

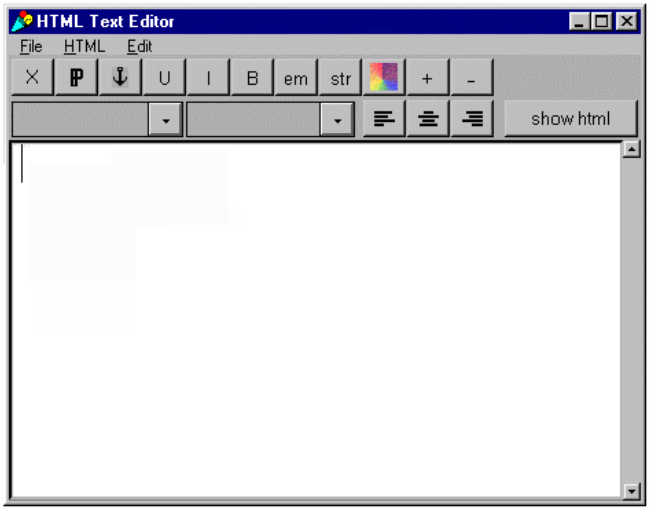
When chosen, enables you to draw filled rectangular areas.

Fill mode:

Fills the region.

HTML Text Editor

Use the HTML Text Editor to create and edit HTML text resources, which can be integrated into an HTML text widget.



This section contains descriptions of:

- Menu Bar Commands
- <Operate> Menu Commands
- Formatting Options
- Editing and Display Controls

Menu Bar Commands

File Menu

Open…

Loads text from a file into the HTML Text Editor. Prompts you for the filename.

Save as…

Saves the HTML text to the specified file. HTML text is saved in ASCII format with HTML markup included.

Exit

Quits the HTML Text Editor.

HTML Menu

New
Clears the HTML Text Editor of the current text and prepares for new text.

Read
Edits HTML text that has been applied to a selected widget.

Load…
Loads an HTML text resource from a specified class and selector into the HTML Text Editor.

Install…
Prompts for the method selector and class name where the HTML text resource is to be stored.

Apply…
Applies the HTML text in the HTML Text Editor to an associated canvas.

Edit Menu

Find…
Finds the specified text.

Replace…
Replaces the specified text with other text.

Undo
Reverses the effects of the last editing action.

Cut
Deletes the selected text and places it into the paste buffer.

Copy
Copies the selected text into the paste buffer.

Paste
Places cut or copied text after the insertion point.

Clear All
Deletes all of the text and resets the attributes.

<Operate> Menu Commands

Below are the <Operate> menu commands for the HTML Text Editor's text view. These commands can be accessed by positioning the mouse pointer over the text view and pressing the <Operate> mouse button.

find…
Finds the specified text.

replace…
Replaces the specified text with other text.

undo
Reverses the effects of the last editing action.

copy
Copies the selected text into the paste buffer.

cut
Deletes the selected text and places it into the paste buffer.

paste
Places cut or copied text after the insertion point.

accept
Installs the HTML text resource.

cancel
Reverts to the last accepted HTML text.

Formatting Options

The HTML Text Editor has two drop-down menus; one for formatting paragraphs, and the other for formatting characters.

Paragraph Formatting

The paragraph-control options for the HTML Text Editor correspond to the six HTML heading levels noted by the `<hn>...</hn>` HTML markup tags.

There is no forced hierarchy in these headings, but for consistency you should use the top level (Heading 1) for main headings, and lower levels for progressively less important ones.

Paragraph formatting is applied to the currently-selected text, making that text a separate paragraph of the given heading level. If no text is selected, a blank paragraph of the given heading level is inserted.

Character Formatting

code sample	<code>...</code>	Example of typed code (usually fixed-width font).
typed text	<kbd>...</kbd>	Text to be marked as keyboard input.
variable	<var>...</var>	A variable name.
sample	<samp>...</samp>	A sequence of literal characters.
definition	<dfn>...</dfn>	The defining instance of a term (often rendered bold or bold italic).
citation	<cite>...</cite>	A citation (typically rendered in italics).
typewriter	<tt>...</tt>	Fixed width typewriter font.
none		Clears all character formatting.

Editing and Display Controls

Removes all HTML markup from selected characters.

Inserts a paragraph break.

Makes the selected text into an anchor. Anchors mark a hypertext link or a destination. In the dialog box, specify:

- Link and a URL to mark the anchor as the start of a link to another place in the current HTML text or to another document or location.
- Anchor and a name to mark the anchor as the possible destination of a link from within the same HTML text or from another document.

Inserts one of the following sets of HTML tags:

```
<a href='''>...</a>
<a name='''>...</a>
```

Link or destination text is shown in blue. Link text is underlined. When the insertion point is within text that is designated with an anchor, the anchor name or link appears in the upper right corner of the HTML Text Editor.

Toggles underline of the selected text by inserting or removing the HTML tags <u>...</u>. Underline may be rendered as slanted in some web browsers.

Toggles italics of the selected text by inserting or removing the HTML tags .... Italics may be rendered as slanted in some web browsers.

Toggles bold of the selected text by inserting or removing the HTML tags .... Bold may not be available in all web browsers.

Toggles emphasis of the selected text by inserting or removing the HTML tags .... Emphasis is represented by italic in many web browsers.

Toggles additional emphasis of the selected text by inserting or removing the HTML tags .... Strong emphasis is displayed as bold in most web browsers.

Increases the font size by a standard increment by inserting HTML tags such as .... Several levels of increase are available.

Decreases the font size by a standard increment by inserting HTML tags such as .... Several levels of decrease are available.

Aligns all of the HTML text to the left.

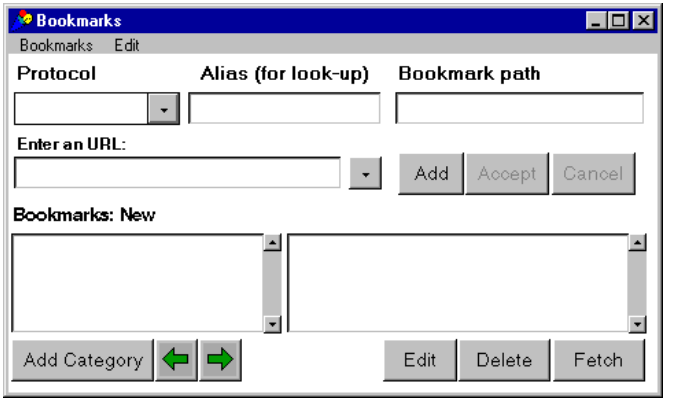
Aligns all of the HTML text in the center.

Aligns all of the HTML text to the right.

Toggles the display in the text view between the HTML text and its HTML markup. This button only affects display in HTML Editor; it does not affect what is stored in the HTML text resource.

Bookmark Manager

Use the Bookmark Manager to create and edit sets of bookmarks.



Menu Bar Commands

Bookmarks

New

Clears the current set of bookmarks from the Bookmark manager and prepares for a new set. You are prompted for a name for the new set.

Load

Loads a set of bookmarks from the current VisualWave image into the Bookmark Manager.

Save

Saves the current set of bookmarks from the Bookmark Manager to the current VisualWave image so that they are available for use within the image. Does not save the VisualWave image; you must do that for the changes to be in effect the next time you start VisualWave.

Load from file…

Loads a set of bookmarks from an external file into the current VisualWave image and into the Bookmark Manager.

Save to file…

Saves the current set of bookmarks to an external file from which they can be loaded into other VisualWave or VisualWave Server images.

Edit

Delete…

Deletes the specified set of bookmarks. All bookmarks in the set are deleted from the current VisualWave image. If the bookmarks also exist in an external file, the file is not affected.

Preferences…

Displays the bookmark preferences:

Cache relative URLs	Causes VisualWave to cache URLs that do not begin with a protocol. For example, the following URL is only cached when this option is chosen: `myGraphics/VWLogo.gif`.
Cache bookmarks in URL list	Causes VisualWave to cache bookmark paths, when they are entered into fields that accept URLs.
Number of URLs to cache?	The number of URLs (including relative URLs and bookmarks when the above options are chosen) that are cached by VisualWave and displayed in menus that offer a list of URLs.
Empty Cache	Clears the cache of URLs.
Apply	Makes changes to preferences take effect.
Close	Closes the preferences window.

<Operate> Menu Commands

Category List

Bookmark categories listed in bold have subcategories. To see the list of subcategories, double-click on the bold category name.

Copy

Copy the selected category and all its bookmarks into the buffer.

Cut

Remove the selected category and all bookmarks within it.

Paste→After

If the buffer contains a category, insert that category and its bookmarks after the selected category. The bookmark paths for all bookmarks within the category are also changed to reflect their new organization. Note that places you have used the bookmark paths are *not* updated automatically; you must update them manually. If no category is selected, this command has no effect.

Paste→Before
If the buffer contains a category, insert that category and its bookmarks before the selected category. The bookmark paths for all bookmarks within the category are also changed to reflect their new organization. Note that places you have used the bookmark paths are *not* updated automatically; you must update them manually. If no category is selected, this command has no effect.

Paste→Indented
If the buffer contains a category, insert that category and its bookmarks within the selected category, as an indented subcategory. The bookmark paths for all bookmarks within the category are also changed to reflect their new organization. Note that places you have used the bookmark paths are *not* updated automatically; you must update them manually. If no category is selected, this command has no effect.

Rename
Change the name of the selected category. The bookmark paths for all bookmarks within that category are also changed to reflect their new organization. Note that places you have used the bookmark paths are *not* updated automatically; you must update them manually.

Add→After
Insert a new empty category after the selected category. If no category is selected, the new category is added at the end of the list. When prompted for a category name, you can enter any string.

Add→Before
Insert a new empty category before the selected category. If no category is selected, the new category is added at the top of the list. When prompted for a category name, you can enter any string.

Add→Indented
Insert a new empty category within the selected category. If no category is selected, the new category is added at the bottom of the list. When prompted for a category name, you can enter any string.

URL List

The URL List appears in the bottom right pane of the Bookmark Manager. For the category selected in the left page, the URL List displays all the URLs for which bookmarks are defined.

Copy URL
Copy the selected URL into the buffer. Once copied, you can paste the URL into any property that accepts an URL.

Copy Path

Copy the bookmark path for the selected URL into the buffer. Once copied, you can paste the path into any property that accepts an URL.

Browse References

Display all of the methods that refer to the selected URL by its bookmark path.

Buttons

Add the specified bookmark to the selected category. If no category is selected, this command has no effect. You must specify a Protocol, Alias, and URL. The Alias cannot be one that is already used in the current category. When you add a bookmark, the Bookmark Manager automatically generates a bookmark path for it, based on the names of the bookmark set, category, and alias. For an example, see "Creating Bookmarks" on page 95.

Causes changes to a bookmark's protocol, URL, or alias to be stored in the Bookmark Manager. Note that the change is not stored in the image and made generally-available until you save the bookmark set by choosing **File→Save**. For an example, see "Example: Updating Bookmarks" on page 100.

Causes changes to a bookmark's protocol, URL, or alias to be dismissed.

Allows you to edit the protocol, URL, or alias for the bookmark selected in the URL list. The bookmark path is automatically-generated and cannot be edited. If you change the alias, however, the bookmark path is automatically updated to reflect the change. Note that places you have used the bookmark paths are *not* updated automatically; you must update them manually.

Note that to make your change take effect, you must do two things: you must click the **Accept** button to store the change in the Bookmark Manager and you must choose **Bookmarks→Save** to store the changed bookmark set in the current VisualWave image.

For an example, see "Example: Updating Bookmarks" on page 100.

Remove the selected bookmark from the selected, category, the current bookmark set, and from the VisualWave image. References to it are not deleted; however, you can find references using the **Browse References** command on the URL List's <Operate> menu.

Retrieve the image or document for the selected bookmark and displays it in a small viewer window. Useful for testing URLs. Applicable only for URLs with the protocol HTTP or relative URLs that can be found by an active TinyHttpServer within the image (usually found using a FileResponder resolver).

Add a new, empty category at the bottom of the list. When prompted for a category name, you can enter any string. Category names must be unique within a bookmark set.

Move the selected subcategory up one level in the hierarchy. The Bookmark Manager also changes the bookmark paths for all of the bookmarks in the category to reflect the new organization. Note that places you have used the bookmark paths are *not* updated automatically; you must update them manually. If no category is selected or if the selected category is already at the top level, this command has no effect.

Move the selected category down one level in the hierarchy, making it a subcategory of the category above it. The Bookmark Manager also changes the bookmark paths for all of the bookmarks in the category to reflect the new organization. Note that places you have used the bookmark paths are *not* updated automatically; you must update them manually. If no category is selected or if the selected category is already a subcategory at the lowest level, this command has no effect.

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®