Internet
Application Guide

SoftVelocity

This publication supports Clarion. It is possible that it may contain technical or typographical errors. SoftVelocity Incorporated provides this publication "as is," without warranty of any kind, either expressed or implied.

SoftVelocity Incorporated
2769 East Atlantic Blvd.
Pompano Beach, Florida 33062
(954) 785-4555
www.softvelocity.com

Trademark Acknowledgements:

SoftVelocity is a trademark of SoftVelocity Incorporated.
Clarion™ is a trademark of SoftVelocity Incorporated.
Btrieve® is a registered trademark of Pervasive Software.
Microsoft®, Windows®, and Visual Basic® are registered trademarks of Microsoft Corporation.
All other products and company names are trademarks of their respective owners.

Printed in the United States of America (0803)

Contents:

10 - The Internet Builder Class Templates 175

Introduction

What is WebBuilder and Internet Connect?

Clarion works together with both Internet Connect and WebBuilder to web-enable database applications so that you can use the same application locally (i.e., under Windows, Windows 95, Windows 98, or Windows NT) or on the Web using any JavaScript enabled browser. Internet Connect also requires a Java enabled browser.

This book is provided to give you an understanding of Clarion's internet technologies. WebBuilder is a Java-free internet development extension to the Clarion development environment. Internet Connect requires Java support. This allows you to create web applications in a product that you already know about. The applications you create can be compiled to run as a desktop Windows application or to run in an internet browser.

WebBuilder creates pure HTML pages dynamically at runtime based on the designed application. This product includes Skeletons which can be customized to fit the look and feel of you applications. Skeletons can be modified without recompiling your application. Along with Skeletons, TSSCRIPT (a scripting language) is also introduced in this product.

The goal of this manual is to get you familiar enough with some of the basics of these two technologies in order to make good decisions about your Web applications.

This book assumes you have completed the tutorials in the Clarion *Getting Started* and *Learning Clarion* . If you have not yet done so, we urge you to do them before gettng started. It is helpful to understand the basic Clarion concepts first. It is also helpful to be familiar with the way Web browsers work. Some basic HTML knowledge is also useful. Provided as a pdf file is a simple intoduction to the HTML language.

Clarion Internet Technologies and the Clarion Development Environment

Automatic application developer for Windows or Web

When you just need a “simple” application to maintain a database, you can literally do the job in minutes using Clarion. The key is the database dictionary. If the Application Generator knows what files or tables you want in the application and how they’re related, it can build an application. So all you need to do is select one or more files then indicate (when there are two or more files) whether the files have a one to many relationship or a many to one relationship.

The Application Wizard can then create a full-featured application, and by merely checking a box on one of the wizard’s dialogs, you can transform the application into a Web-enabled application. The resulting application can run locally or on the Web using the Clarion Application Broker.

Visual development environment for Windows or Web

With Clarion, dropping a control in a window gives you a lot more than other Rapid Application Development tools. These tools typically let you add a user interface control, but then expect you to write the code to implement its functionality. With Clarion, you add a template, which contains the control, data, and executable code. That means you don’t have to write code—one click places a complete business solution: a user interface control and the code that enables it to do its job. Moreover, each template has its own user interface. When you view the properties for the template, you’ll see an “Actions” tab. By checking a box, choosing a dropdown list item, or filling in an edit box, you can customize the behavior of the template so that it meets your needs exactly. You’ll set “Actions” for the templates at many places in the longer tutorial in this book.

When you use the template interface to specify these behaviors, the Application Generator writes the code (Clarion language source code) that implements the behavior for you. Using the templates, you can do an awful lot of custom programming without writing a single line of source code.

This paradigm extends to the web implementation of your application. All of the underlying functionality is transformed to represent your application inside a browser. Concurrency checking and referential integrity are automatic in your application and are enforced over the web in a similar manner. Additional Internet Options allow you to control event handling so that you can specify the conditions under which an event is processed on the server.

What You'll Find in this Book

The following lists the chapters of this book and summarizes its content:

Part I—WebBuilder

Making a Web-enabled application
Chapter One: This chapter covers how to web-enable an application. It leads you through the process step by step. Some deployment steps are also covered so you can test your web-enabled applications.

Differences between Web and Windows applications
Chapter Two: This chapter discusses the placement of controls in a Windows application. It also covers the difference in static vs. dynamic HTML. This chapter also introduces you to skeletons and what they are.

Web Template Guide
Chapter Three: This chapter documents the Web templates.
TSSCRIPT
Chapter Four: This chapter introduces TSSCRIPT, the scripting language that is used to create runtime HTML pages.

Skeleton Guide
Chapter Five: This chapter provides a reference to the skeleton files. It explains each skeleton and it's purpose.

Common Questions
Chapter Six: This is the chapter where everything comes together. These questions have been gathered from several sources including the newsgroups. A solution is provided with each question.

Part II—Internet Connect

Application Wizard Tutorial
Chapter Seven: A few quick steps with the Application Wizard allow you create to a complete web application in five minutes.

Web-enabling an Existing Application
Chapter Eight: Using the IBC templates to port Clarion applications to the Web.

Advanced Web Programming Techniques
Chapter Nine: Introduces the customization capabilities offered by the IBC templates. It walks you through modifying your application for optimal performance and functionality on the web.

Using the Internet Builder Class (IBC) Templates
Chapter Ten: A reference to the IBC Template interface.

Application Design Considerations
Chapter Eleven: Tips and techniques on web-based application design.

Internet Builder Class Library- A Quick Reference
Chapter Twelve: A quick guide to the template implementation of the objects in the Internet Builder Class (IBC) Library. This chapter lists properties and methods commonly used in web-based applications.

Glossary
Glossary of terms

The PDF versions of all manuals are indexed to allow fast searches across all manuals (requires Acrobat Reader with Search).

Where to Find More Information

The *Application Broker* manual is the guide to installing, configuring, and using the Clarion Application Broker.

The PDF versions of the manuals are indexed to allow fast searches across all manuals (requires Acrobat Reader 3.x with Search; the installation program is on the CD).

Important: if any part of the online help text conflicts with the printed documentation, the information in online help should take precedence. SoftVelocity makes every reasonable effort to ensure the printed documentation is up to date. However, the lead-time required by printers may create a lag in the documentation; while we can update the online files that ship concurrently with a product revision, printed materials must “catch up” later.

Documentation Conventions

Typeface Conventions

Italics	Indicates what to type at the keyboard, such as *Enter This*. It is also used to identify the title bar text of a window.
CAPS	Indicates keystrokes to enter at the keyboard, such as ENTER or ESCAPE, or to CLICK the mouse.
Boldface	Indicates prompts or options from a pulldown menu or text in a dialog window.
`Courier New`	Used for diagrams, source code listings, to annotate examples, and for examples of the usage of source statements.

Keyboard Conventions

F1	Indicates a single keystroke. In this case, press and release the f1 key.
ALT+X	Indicates a combination of keystrokes. In this case, hold down the ALT key and press the X key, then release both keys.

Product Information

Registering This Product

Before you begin using your Clarion internet product, be sure to fill out and mail in the registration card that came in the package. This Business Reply Card makes you eligible to receive several important benefits. Once registered, you can use SoftVelocity's Technical Support services and you automatically receive new product announcements and update alerts.

Technical Support

Help can be obtained from several different online newsgroups. Our web site, www.softvelocity.com, details the available technical support plans.

Usenet Newsgroup--comp.lang.clarion

You can participate in the Clarion Usenet Newsgroup on the Internet--comp.lang.clarion. In this newsgroup, Clarion programmers from around the world exchange ideas and techniques. Log into your News Server and subscribe to comp.lang.clarion. If your news server does not carry the feed, you should contact your Internet provider.

SoftVelocity's product newsgroups

SoftVelocity's internal newsserver offers newsgroups for all SoftVelocity products. To subscribe to these groups use news.softvelocity.com as the news server. There are several newsgroups you can subscribe to on this server.

SoftVelocity's Web Site:

You can find other Clarion resources on the Internet by visiting SoftVelocity's site on the World Wide Web:

http://www.softvelocity.com

Paid Technical Support

Paid telephone technical support is available. Refer to the SoftVelocity web site for the most up to date information on the available technical support plans.

Part I

WebBuilder Technology

1 - Web-enable an Example Application

Introduction

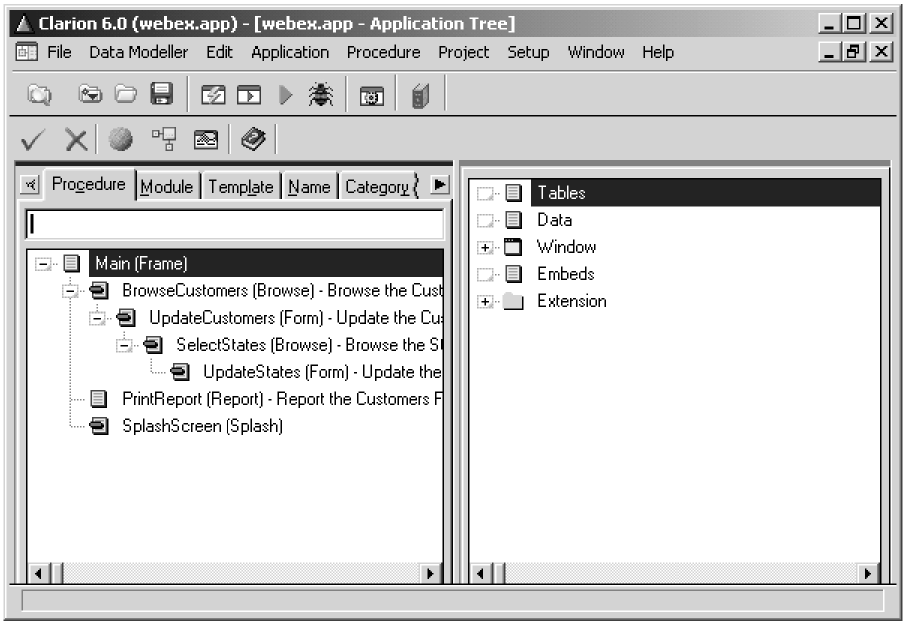
This chapter goes through an example web-enabled application like an annotated example.

It covers the templates used, which settings were used and why. It also covers running the application and what you should be seeing when you do.

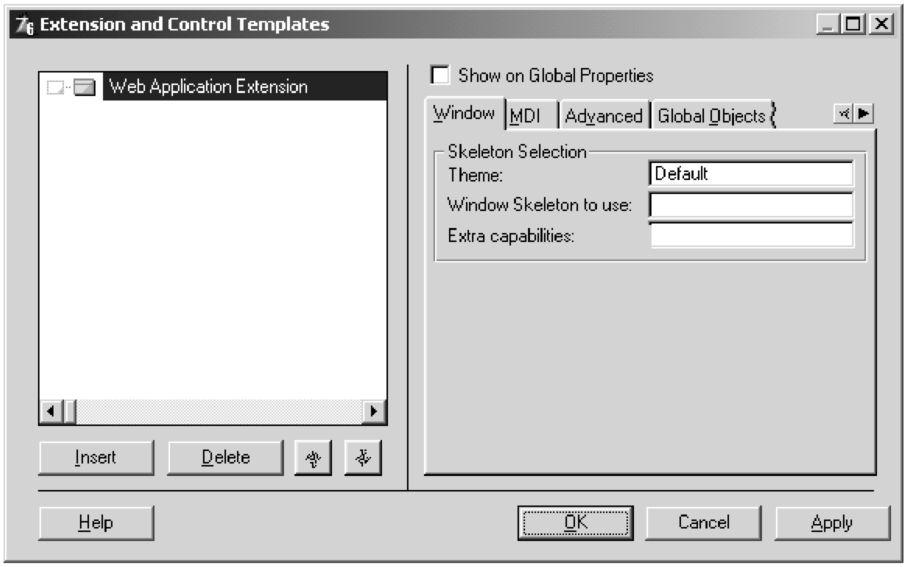
Upon running the application, some areas may not look right. How does one fix them? This part covers the skeletons, how to change themes and walks through a small skeleton to show what it does. We will also look at the generated HTML code in your internet browser while the application is running.

Starting Point

Start Clarion. Open the example Web application, located in the *\Clarion6\Examples\Webex\* folder. Your desktop should look like this:


Clarion 6.0 (webex.app) - [webex.app - Application Tree]
File
Data Modeller
Edit
Application
Procedure
Project
Setup
Window
Help
Procedure
Module
Template
Name
Category
Main (Frame)
BrowseCustomers (Browse) - Browse the Cust
UpdateCustomers (Form) - Update the Cu
SelectStates (Browse) - Browse the St
UpdateStates (Form) - Update the
PrintReport (Report) - Report the Customers F
SplashScreen (Splash)
Tables
Data
Window
Embeds
Extension

The first stop is the Global button, and the **Extensions** button therein. This is the place to set all the defaults for the application. You will see the *Web Application Extension* highlighted. This extension is required for all web applications.



☐ Show on Global Properties

Skeleton Selection

Theme: Default

Window Skeleton to use:

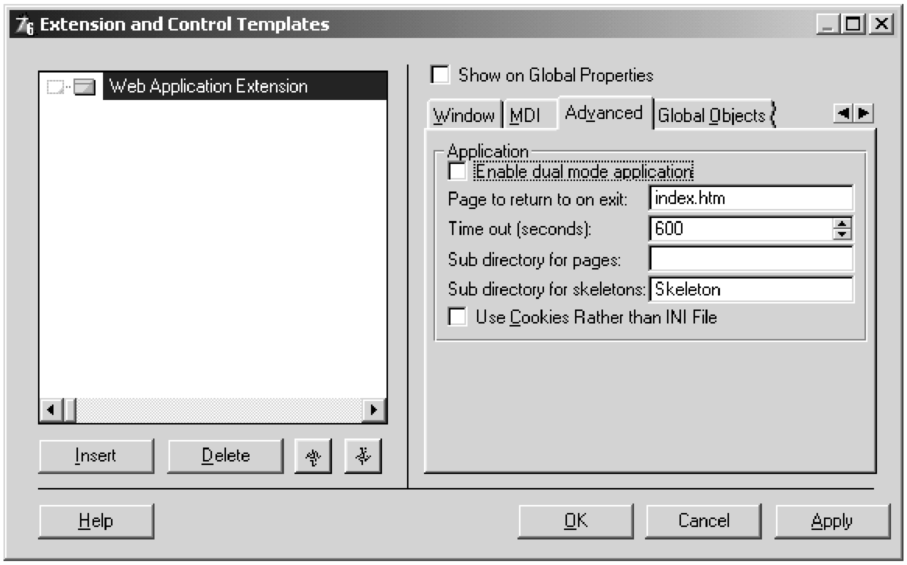
Extra capabilities:

This is the dialog where global extension templates are added to the application. As shipped, there are 3 themes for your Web applications, the default is used here.

Note:

Detailed information about the template dialogs is in the Web Template chapter.

Select the **Advanced** tab.



☐ Show on Global Properties

Application

☐ Enable dual mode application

Page to return to on exit: index.htm

Time out (seconds): 600

Sub directory for pages:

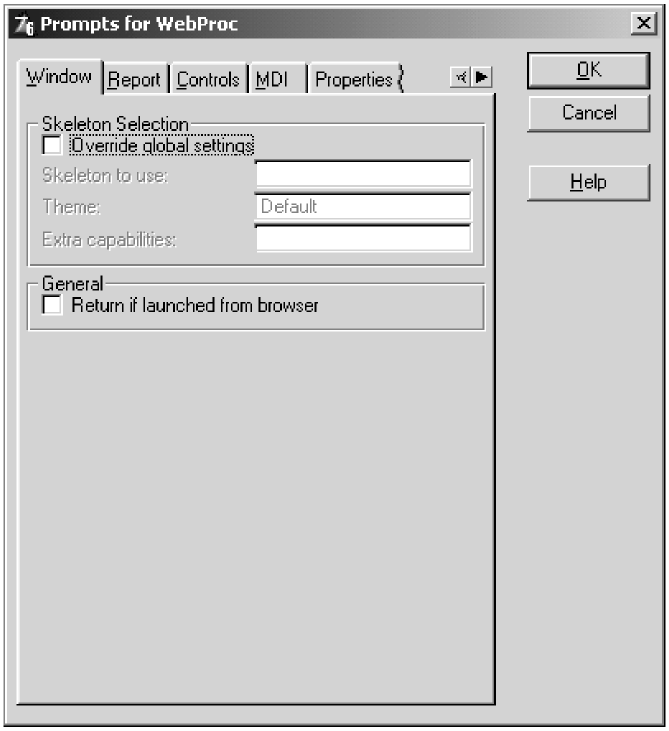
Sub directory for skeletons: Skeleton

☐ Use Cookies Rather than INI File

This dialog shows the time out value. The timeout value means if there is no activity (like a keystroke) detected in the specified number of seconds, the application will automatically terminate. Depending on your use, you may adjust this setting to a higher or lower value.

Press the **OK** button until you return to the application tree. If you add this global extension to an existing application, it causes a procedure extension template to be added to every procedure in your application, with the defaults.

Highlight the *Main* procedure. On the right hand of the Clarion desktop, expand the *Extension* tree, if it is collapsed. You do this by clicking on the plus sign. You will see a Web Procedure Extension template entry. RIGHT-CLICK on it and then choose **Properties** from the popup menu. You should see this dialog:



Skeleton Selection

☐ Override global settings

Skeleton to use:

Theme: Default

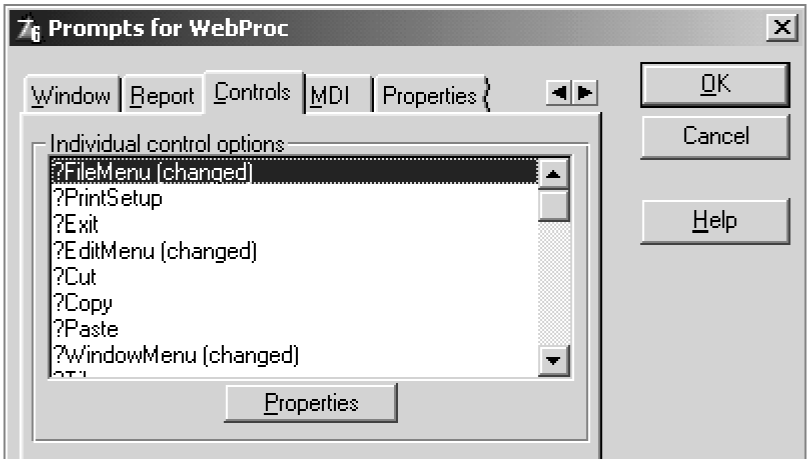
Extra capabilities:

General

☐ Return if launched from browser

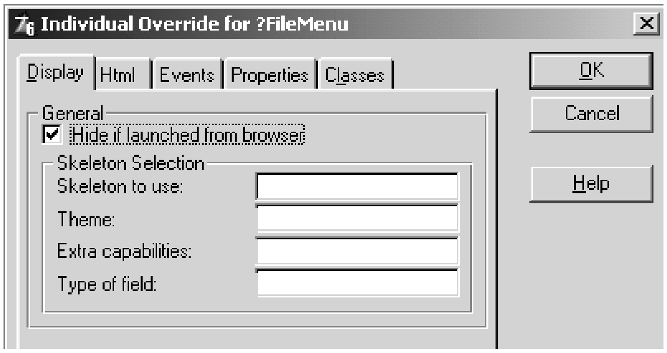
This dialog, the Web Procedure Extension, is similar to the global Web Application Extension. There are some template prompts that allow your application to have desktop specific vs. Web specific functionality.

If you recall, Clarion builds a default menu for you. Some of these menus should never be seen by a Web application. Choose the **Controls** tab.



?FileMenu (changed)
?PrintSetup
?Exit
?EditMenu (changed)
?Cut
?Copy
?Paste
?WindowMenu (changed)

You can see that some of the controls are changed from the defaults (which is to include everything from a desktop application). Press the **Properties** button for the first item in this list, *?FileMenu*. The following dialog appears:



☑ Hide if launched from browser

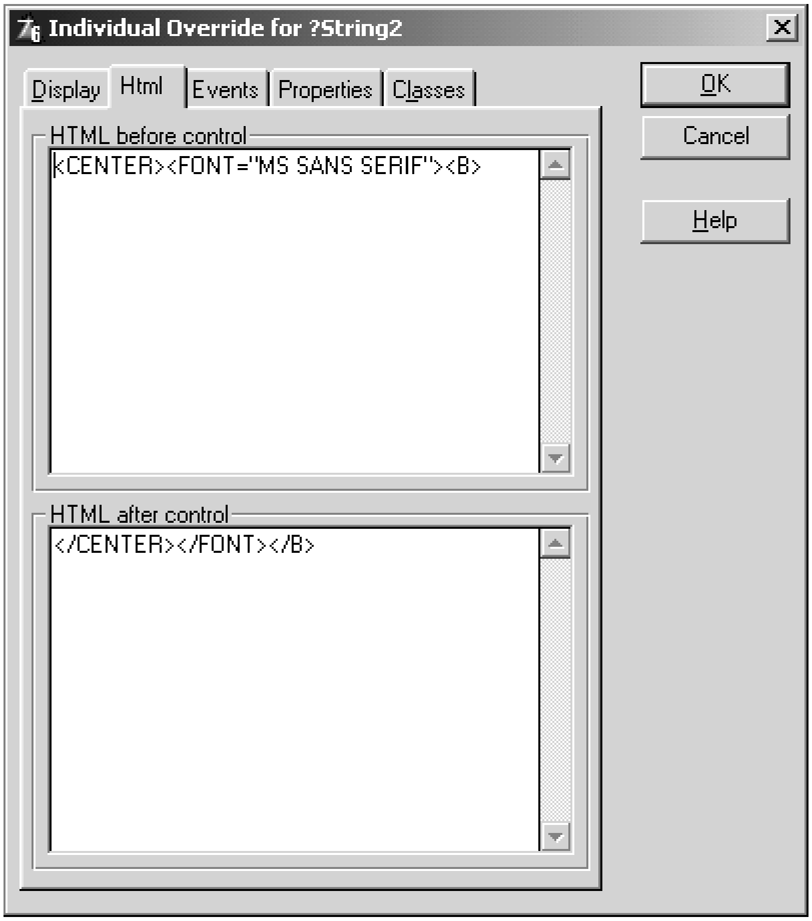
Skeleton to use:
Theme:
Extra capabilities:
Type of field:

The check box, **Hide if launched from browser** means that when this application is launched as a Web application, the menu is hidden. It is visible if run as a desktop application.

This is used because the normal “File” menu is not applicable when running in a browser. There are other menu items that have also been changed. If you look closer, you see the menus are not changed. This is not needed as hiding a menu will hide all items in it. The other menus that should be hidden are the “Edit” menu (where you normally find Cut, Copy, Paste, etc) and the “Window” menu (where you find the Tile, Cascade, and list of open windows, etc).

Press the **OK** button until you are back at the application tree.

Select the *Splash* procedure at the bottom of the tree. Open the Web Procedure Extension template like you did previously. Choose the **Controls** tab. Press the **Properties** button for *?String2*, then choose the **HTML** tab. You will see this dialog:



HTML before control

```
<CENTER><FONT="MS SANS SERIF"><B>
```

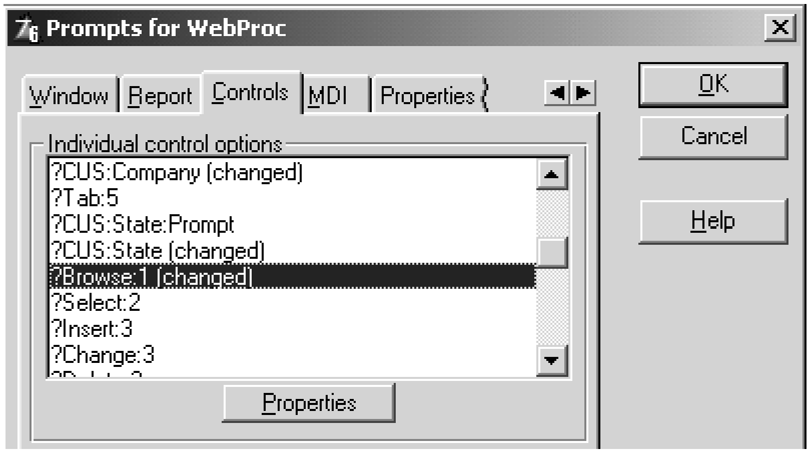
HTML after control

```
</CENTER></FONT></B>
```

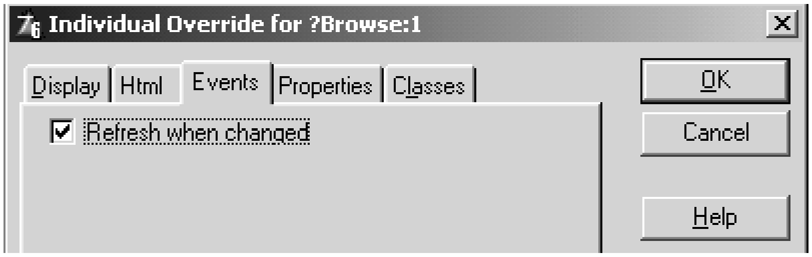
This is where you can add static HTML code before and after the control. In this case, center the text, change to MS Sans Serif font with the bold attribute. The after control text box are the required ending HTML tags. If the end tags were not entered here, then every control appearing afterwards will inherit these changes. This is usually not the desired effect.

The other changed controls are the same. Press the **OK** or **Cancel** button to return to the application tree.

Select the *BrowseCustomers* procedure and open its Web Procedure Extension template. Choose the **Controls** tab. Scroll down until you see *?Browse:1 (Changed)*.



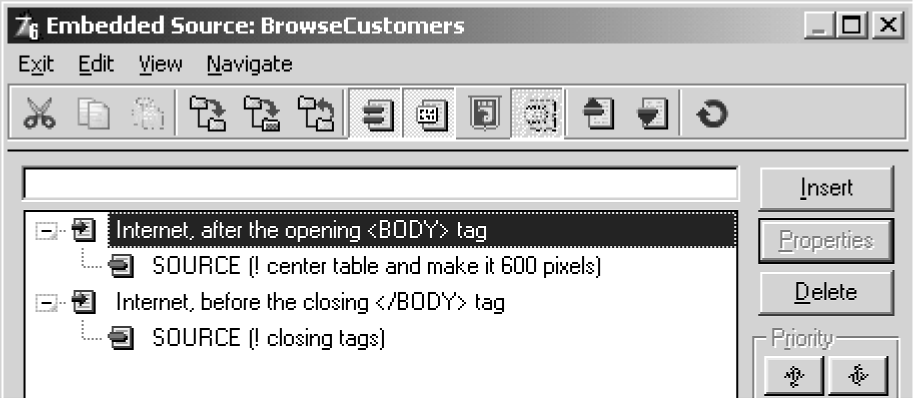
Open its properties and choose the **Events** tab. You will see this dialog:



This is a powerful feature for Web applications. When any column in a listbox changes, the page is refreshed. This means that the listbox will always display the correct values in the browser. Close the dialog and look at the other changed controls. They also have this box checked. This is because of the strings on the window of the procedure.

Close all dialogs until you come back to the application tree. Now let's look at a way of embedding HTML code in embed points.

RIGHT-CLICK on the BrowseCustomers procedure and choose **Embeds**. Press the **Show Filled Only** and **Expand Filled** icon buttons on the embed toolbar. You should see something similar to this:



Internet, after the opening <BODY> tag
SOURCE (! center table and make it 600 pixels)
Internet, before the closing </BODY> tag
SOURCE (! closing tags)

The two Internet only embed points are clearly visible. The first embed, *Internet - after the opening <BODY> tag* has this code in it:

```
! center table and make it 600 pixels
TARGET.Writeln('<div align="center"><center>')
TARGET.Writeln('<Table width="600"><TR><TD>')
```

Writeln is a method that sends text into an open document. What is happening here is that we want the list box centered in the browser and with a restricted width so it looks like the desktop version. This embed is not required for the functioning of the application, but it clearly looks better.

The next embed, *Internet - before the closing </BODY> tag* has this source:

```
! closing tags
TARGET.Writeln('</td></tr></Table>')
TARGET.Writeln('</center></div>')
```

Since the tags in the first embeds require closing tags, the same technique is used to insert them into the HTML generated page, at the proper point.

Press the **OK** or **Cancel** buttons until you return to the application tree.

Feel free to look at other procedures and their settings. The Invoice application is designed to run with the linked-in application broker (the app runs in the default browser).

When you are running the application, do some order edits and see how it acts in a browser. The areas covered here should now be obvious in their effects.

These areas are not the only way to accomplish the desired effect. There are many other ways. The rest of this book covers some alternate methods. As you may have come to expect from using Clarion tools, there is seldom a single correct way to do a task!

2 - Web vs Windows Applications

Introduction

This chapter asks and answers some basic questions as well as introduces the concept of skeletons, the scripting language used in the skeletons and how you could use it.

Also, how does one use HTML in Clarion applications? Must one use the skeletons? What is the role of skeletons and what is TSSCRIPT? How does the broker fit in?

What is a Skeleton?

Simply put, a skeleton is an HTML file with a scripting language embedded in them. The role of these scripts is to take the window controls and their attributes and dynamically merge them with the skeleton to generate the correct and functioning HTML code at runtime.

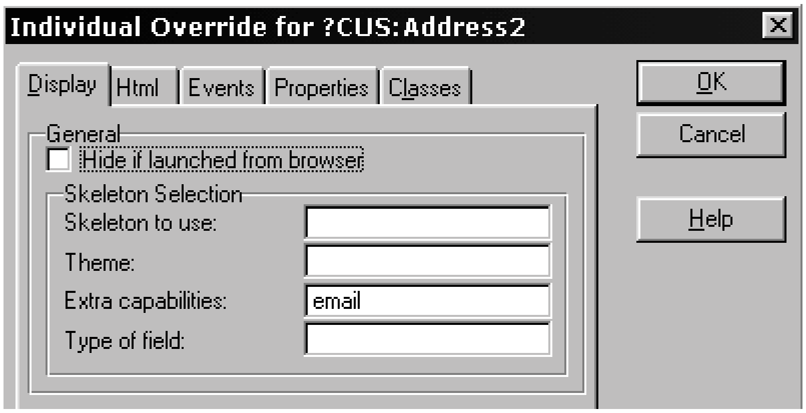
What is TSSCRIPT?

TSSCRIPT is the scripting language used in the skeletons. The scripts themselves are useful, but in the traditional Clarion style, there are underlying objects with template interfaces.

Here is an example. Suppose you want to use a column in a table that stores a customer's email address? You can take advantage of TSSCRIPT to accomplish this. First, here is the interesting bits from the email skeleton (hotstring.htm):

```
<meta name="ts-control" content="sstring">
<meta name="ts-capabilities" content="email">
</head>
<BODY>
<!-- HotString.htm -- Start -->
```

Notice the *ts-capabilites* and the *content*. Now, let's inspect the template dialog for a individual control override:

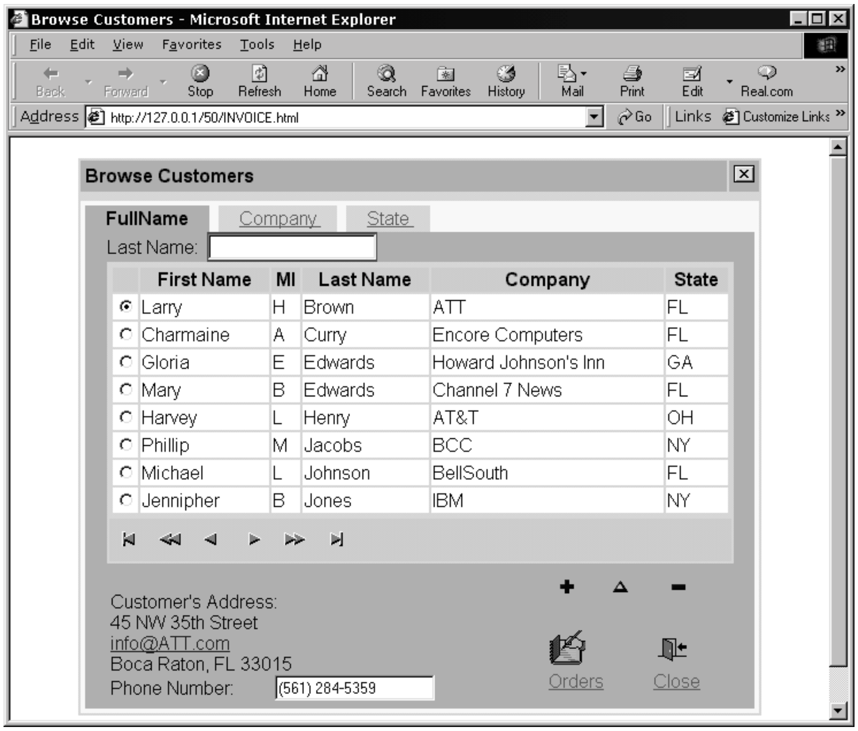


☐ Hide if launched from browser

Skeleton to use:	
Theme:	
Extra capabilities:	email
Type of field:	

There is a template prompt and a value to set in order to apply this to a control. You should also notice that the *ts-control* defines the content of the email control to be a STRING (the type of string is unimportant).

Simply place the string control on a window (like a browse procedure). When you run the application in a browser, it will look like this:



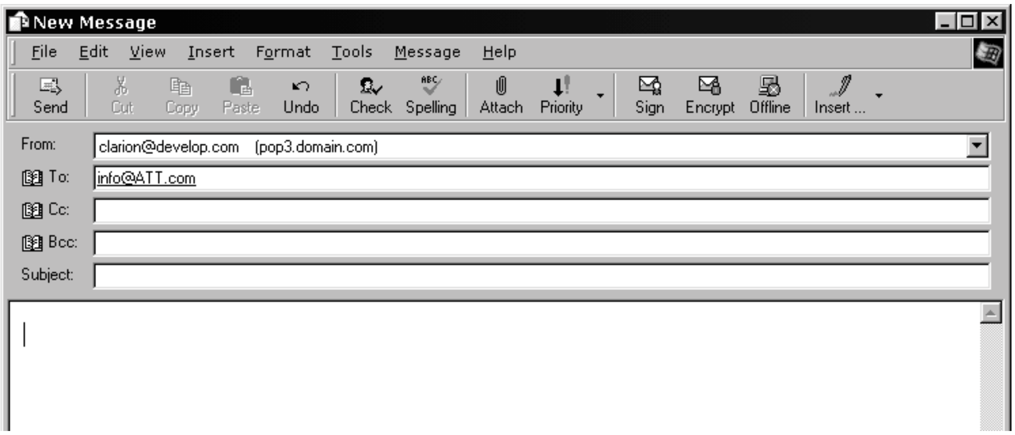
Address: http://127.0.0.1/50/INVOICE.html

Last Name:

	First Name	MI	Last Name	Company	State
●	Larry	H	Brown	ATT	FL
○	Charmaine	A	Curry	Encore Computers	FL
○	Gloria	E	Edwards	Howard Johnson's Inn	GA
○	Mary	B	Edwards	Channel 7 News	FL
○	Harvey	L	Henry	AT&T	OH
○	Phillip	M	Jacobs	BCC	NY
○	Michael	L	Johnson	BellSouth	FL
○	Jennipher	B	Jones	IBM	NY

Customer's Address:
45 NW 35th Street
info@ATT.com
Boca Raton, FL 33015
Phone Number: (561) 284-5359

And if you click on the link, your default email client is launched:



From: clarion@develop.com (pop3.domain.com)
To: info@ATT.com
Cc:
Bcc:
Subject:

The generated HTML looks like this:

```
<!-- HotString.htm -- Start -->
  <A HREF='mailto:info@ATT.com'>info@ATT.com
  </A>
```

This is one way to make a very simple change via the templates, with no embedded code to get the desired feature.

So where do the skeletons come in? The skeletons are covered in detail in a later chapter, but here is how this works. We'll start where we left off in the skeleton:

```
<!-- HotString.htm -- Start -->
<TSSCRIPT tag=a attr=href replace=NAME value=Contents>
  <A HREF="mailto:NAME">
    <TSSCRIPT value=Contents>
    </TSSCRIPT>
  </A>
</TSSCRIPT>
```

Compare the above with what was generated at runtime. You can start to see how TSSCRIPT works. The first line is says that it needs an anchor tag (that is the tag=a attribute), and the anchor tag has an attribute of HREF. It also declares the replace variable, called NAME and a value variable called CONENTS.

The next line is psuedo-HTML code for an email anchor. The actual replacing is done in the next TSSCRIPT line, where it parses the whatever the value of the Contents is. The rest of the lines are the required end tags.

So you have the correct anchor tag for the email generated as shown above.

Dynamic vs. Static HTML

You can use either one you feel fits the need, but best results can be achieved when you use both.

What if you have a column in one of your tables that stores a customer’s email address (as long as we are on this theme), and you want to display this email address as you scroll through the rows on your browse list. But even better, suppose as you scroll through your list, you can simply click on the displayed string to start your email client.

You have a local string variable called DisplayString and it is somewhere on your browse window (not in the list itself). You would want this populated with a “friendly name”, like “Joe Q. Smith”. However, you want this to appear as a link and if you click on it, an email is started.

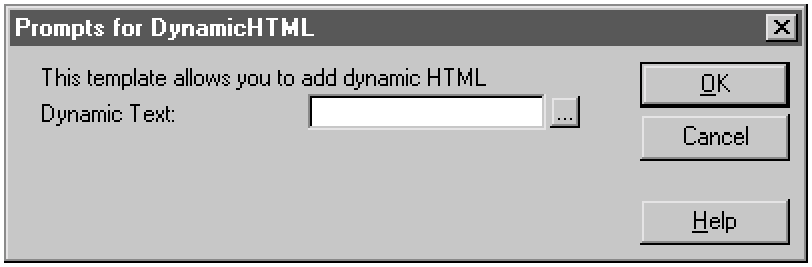
The local variable, DisplayString is populated everytime a new selection in the list is made. In other words, as you scroll up and down the list.the SetQueueRecord embed is used. You could code something like this:

```
DisplayString = 'Reply to ' & CLIP(CUS:FirstName) & CUS:LastName
```

This ensures the string has the proper data visible. However, what we want is to make an email anchor. This is done with HTML code, but some of this needs to be dynamic, like the email address of the person we are sending email to.

In this case, you can use the DynamicHTML code template. You want to put it before the control of *?DisplayString*. There is an embed for that (as well as all controls populated on a window).

The Dynamic HTML code template looks like this:



This template allows you to add dynamic HTML

Dynamic Text:

You could enter the following into this template:

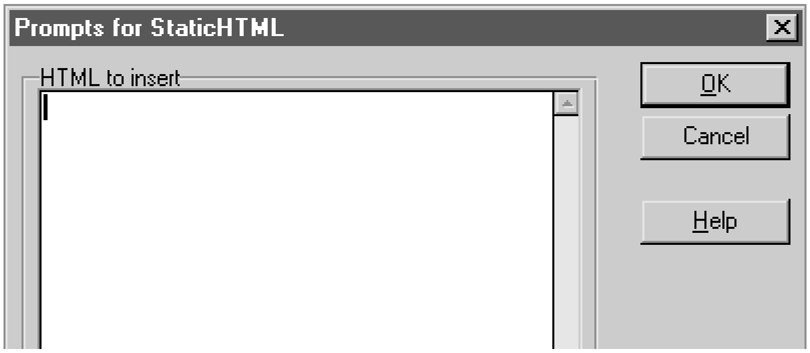
```
'<<a href="mailto:' & CLIP(CUS:Email) & '?Subject=The next DevCon">'
```

Tip

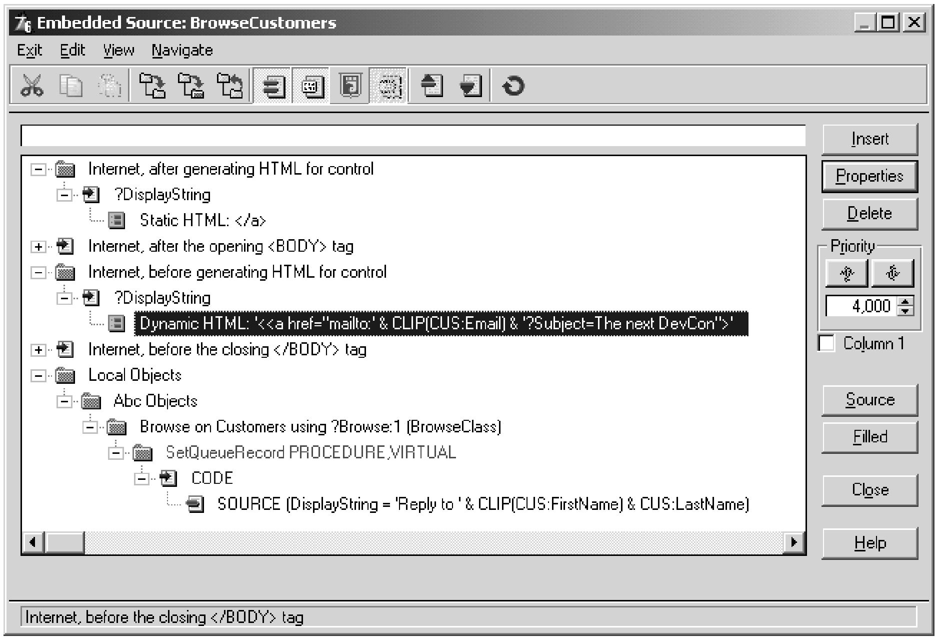
Pressing the ellipsis button will open the variable selection dialog. The use of this lookup is not required.

The next step is to add static HTML code as you will need some code that will never change. This is ideal for end tags. Simply find the Internet embed after generating HTML for the control.

Again, you can use a code template, in this case the StaticHTML code template. Just enter the HTML code you wish to insert after the control.



When you are done, your embed tree will look similar to this:



```
Internet, after generating HTML for control
  ?DisplayString
    Static HTML: </a>
Internet, after the opening <BODY> tag
Internet, before generating HTML for control
  ?DisplayString
    Dynamic HTML: '<<a href="mailto:' & CLIP(CUS:Email) & '?Subject=The next DevCon">'
Internet, before the closing </BODY> tag
Local Objects
  Abc Objects
    Browse on Customers using ?Browse:1 (BrowseClass)
      SetQueueRecord PROCEDURE,VIRTUAL
        CODE
          SOURCE (DisplayString = 'Reply to ' & CLIP(CUS:FirstName) & CUS:LastName)
```

Internet, before the closing </BODY> tag

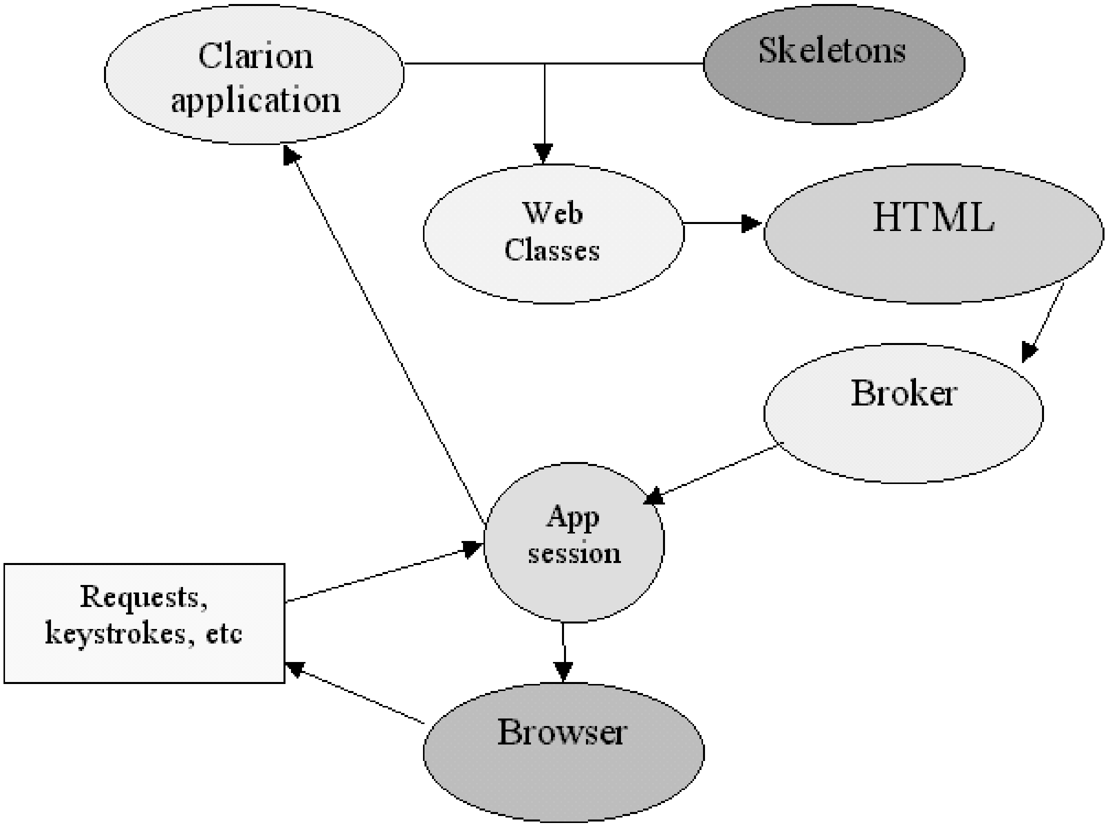
Now you have your link. There are other possiblities you can use with these templates and skeletons.

The Application Broker

There are two forms of the Application Broker. The linked in (executable) broker is used for testing your web developed application. This is automagically linked in when you compile and run an application with the Web templates.

For more details about the Application Broker and various deployment steps, see the Application Broker manual. What is the Applicaiton Broker? What does it do?

Examine the following diagram:

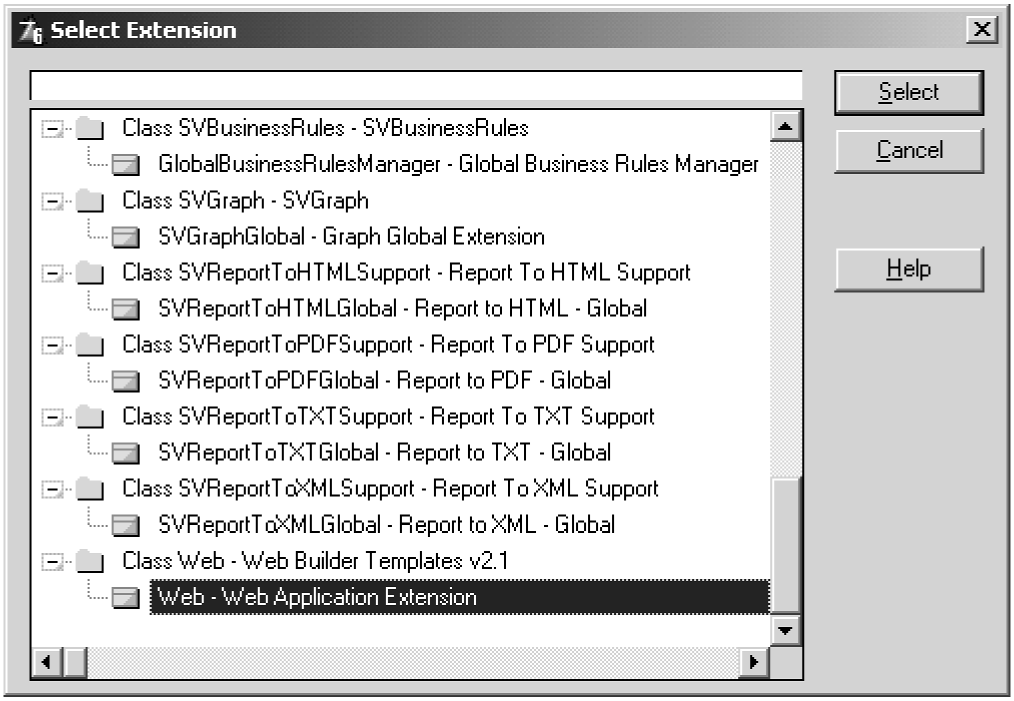


This shows that the broker gets its data from two sources, the Clarion application and the skeletons. It then passes data (HTML pages) to a browser so the user can interact with the program.

3 - *Web Templates*

Web Application Extension

The Web Application Extension is a global template that Web-enables a Clarion application. It adds the functionality of generating dynamic HTML when the application is accessed through the Application Broker. This template allows you to specify the options to use when generating an HTML representation of your windows and reports.


Select Extension
Class SVBusinessRules - SVBusinessRules
GlobalBusinessRulesManager - Global Business Rules Manager
Class SVGraph - SVGraph
SVGraphGlobal - Graph Global Extension
Class SVReportToHTMLSupport - Report To HTML Support
SVReportToHTMLGlobal - Report to HTML - Global
Class SVReportToPDFSupport - Report To PDF Support
SVReportToPDFGlobal - Report to PDF - Global
Class SVReportToTXTSupport - Report To TXT Support
SVReportToTXTGlobal - Report to TXT - Global
Class SVReportToXMLSupport - Report To XML Support
SVReportToXMLGlobal - Report to XML - Global
Class Web - Web Builder Templates v2.1
Web - Web Application Extension
Select
Cancel
Help

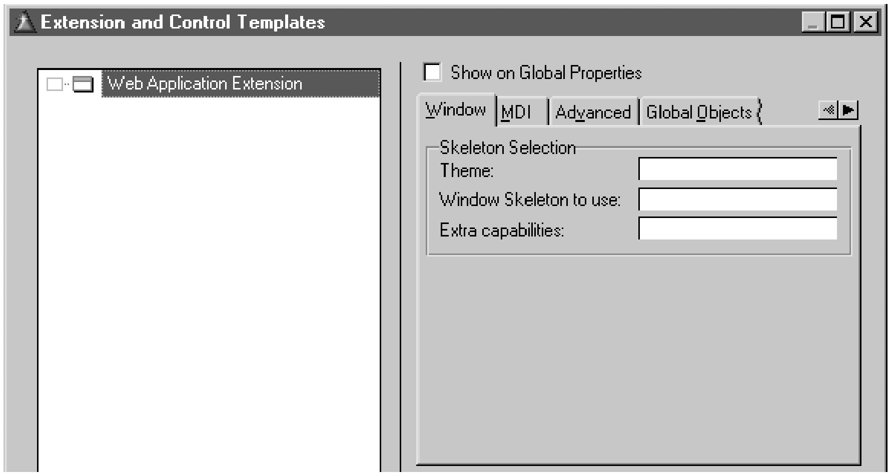
In addition, it automatically adds the Web Procedure Extension to every existing procedure in your application and any procedures subsequently added to the application. The Web Procedure Extension allows you to override many of the global options for a specific procedure.

This template allows you to customize the global appearance and behavior of your application when it is executed over the Web. The settings you specify here are global in nature; that is, they affect every procedure in your application.

You can override most of these settings on a procedure level using the Web Procedure Extension's settings. In addition, some options can be specified on a control-by-control basis. The combination of these three levels of customization provides you with complete flexibility of design.

Window Settings

Skeletons are a collection of HTML files that contain all the information needed to control the construction of the delivered HTML pages. These files consist of true HTML code along with the TSSCRIPT scripting language.



The Window tab allows the global setting of the skeleton to be used for the basic window design of your application’s windows.

Theme

Skeleton files can be categorized into common themes or styles so all window representations in a theme have a common look and feel. Specify the default window theme here.

Window Skeleton to use

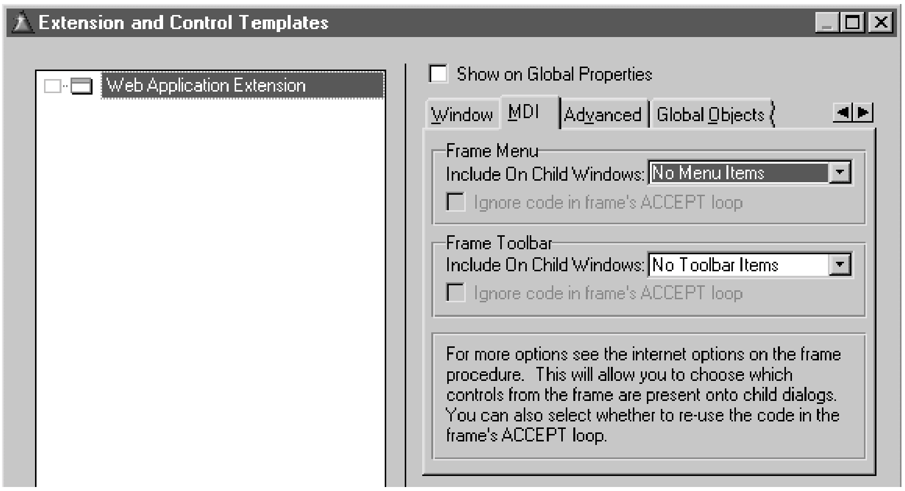
Specifies the default window skeleton to use. This is normally a modified version of WINDOW.HTM skeleton from the supplied skeleton files.

Extra capabilities

Specifies extra capabilities of one skeleton versus another. The capability is specified here.

MDI Settings

This section determines the manner in which Application Menus and Toolbars are handled.



☐ Show on Global Properties

Frame Menu
Include On Child Windows: No Menu Items
☐ Ignore code in frame's ACCEPT loop

Frame Toolbar
Include On Child Windows: No Toolbar Items
☐ Ignore code in frame's ACCEPT loop

For more options see the internet options on the frame procedure. This will allow you to choose which controls from the frame are present onto child dialogs. You can also select whether to re-use the code in the frame's ACCEPT loop.

Tip

For control over specific Menu or Toolbar items, set the MDI overrides in the Frame Procedure's Internet Options.

Frame Menu

This section determines the manner in which Application Menus are handled. This allows you to specify which global menu options are displayed on "child" windows.

Include on Child Windows

Select an option from the drop-down list. The choices are:

All Menu Items	All menu choices appear on child windows.
No Menu Items	No menu choices appear on child windows.

Ignore code in frame's ACCEPT loop

Check this box to ignore any code in the Application Frame's ACCEPT loop for menu items. If not checked, any embedded code implemented in the Frame's ACCEPT loop is automatically implemented in the child procedure.

Frame Toolbar

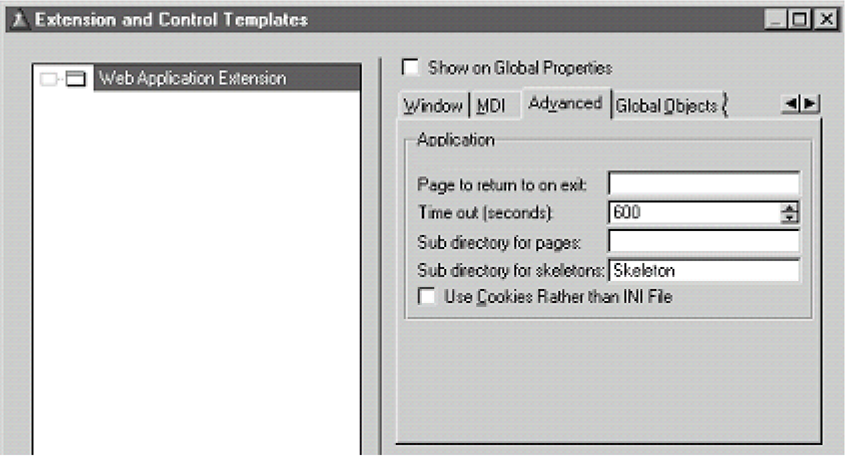
This section determines the manner in which Application Toolbar controls are handled. This allows you to specify which global Toolbar controls are displayed on “child” windows.

Include on Child Windows
Select an option from the drop-down list. The choices are:

> **All Toolbar Items**
> All Toolbar items appear on child windows.
>
> **Standard Toolbar Only**
> Only the Standard Toolbar items appear on child windows. These are the buttons added by the FrameBrowseControl template.
>
> **No Toolbar Items**
> No Toolbar items appear on child windows.

Ignore code in frame’s ACCEPT loop
Check this box to ignore any code in the Application Frame’s ACCEPT loop for toolbar items. If not checked, any embedded code implemented in the Frame’s ACCEPT loop is automatically implemented in the child procedure.

Advanced tab



☐ Show on Global Properties

Page to return to on exit:
Time out (seconds): 600
Sub directory for pages:
Sub directory for skeletons: Skeleton
☐ Use Cookies Rather than INI File

Page to return to on exit
Optionally, specify the HTML page to return to when the program ends. The template generated code calls the WebServer.Init method to set the WebServer.PagetoReturnTo property.

Time out (seconds)
This specifies the maximum amount of idle time (measured in seconds) before an application closes. The default is 600 seconds (10 minutes). The template generated code calls the WebServer.Init method to set the WebServer.TimeOut property.

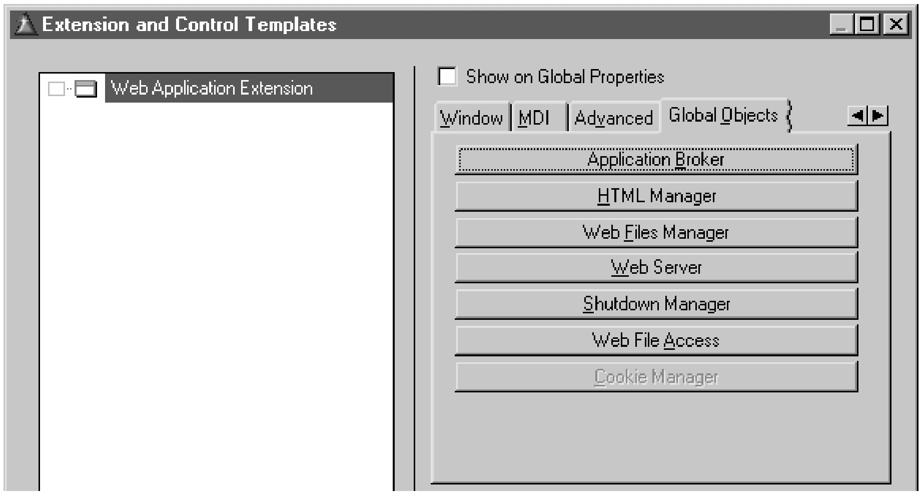
Sub directory for pages
The directory in which the application creates temporary directories (a temporary directory is made for each active connection) to write the dynamic HTML and graphic files. This is also the directory in which to deploy graphic files. If you provide a graphic in this directory, it is not extracted and written to the temporary directory. This defaults to /PUBLIC. The template generated code calls the WebFilesManager.Init method to set the property. It is not appropriate to set this property at runtime.

Sub directory for skeletons
The directory in which the application skeleton files are stored. This defaults to SKELETON. The skeletons must be available at runtime. Multiple directories may be specified. They are separated by a semicolon (;). The template generated code calls the AddSkeletonDirectory method to set the path.

Use Cookies Rather than INI File
Check this box to use cookie files instead of an INI file for storage of data related to a web site.

Global Objects tab

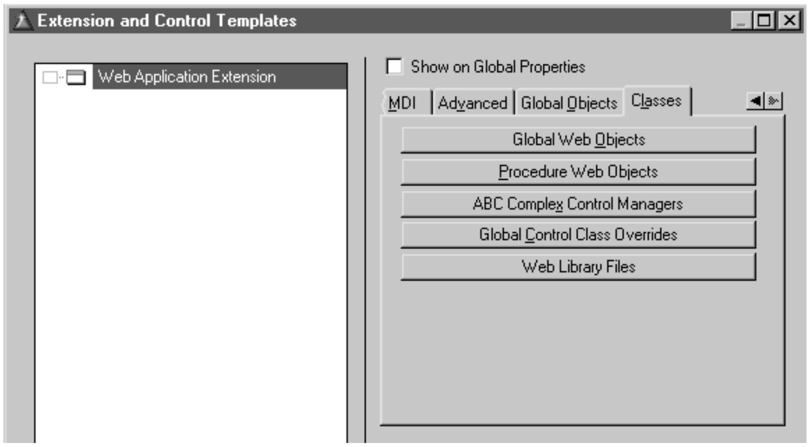
The Global Objects tab lets you specify which classes (objects) the templates instantiate globally in your application to accomplish various tasks, and the source modules that contain the class definitions. This approach gives you the capability to use as much of the WBC Library as you want and as much of your own classes as you want.



To change the class for an item or override the class, press the button for the class you wish to affect.

Classes tab

The Classes Tab lets you specify which classes (objects) the templates use to accomplish various tasks, and the source modules that contain the class definitions. This approach gives you the capability to use as much of the WBC Library as you want and as much of your own classes as you want.


Extension and Control Templates
Web Application Extension
☐ Show on Global Properties
MDI
Advanced
Global Objects
Classes
Global Web Objects
Procedure Web Objects
ABC Complex Control Managers
Global Control Class Overrides
Web Library Files

To change the class for an item or override the class, highlight it in the list, then press the Properties button.

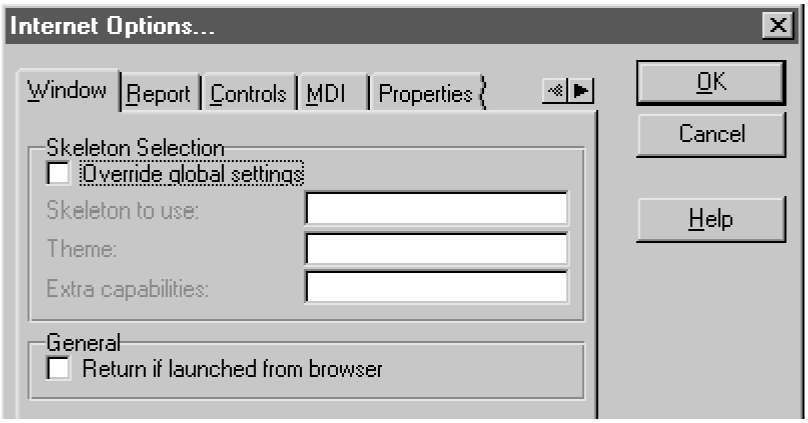
Web Procedure Extension

This template allows you to customize the appearance and behavior of a procedure when it is executed over the Web. The settings you specify here are local in nature, that is they affect only this procedure. To change Global Settings: press the **Global** Icon Button on the Application Generator, then press the **Extensions** button, and modify the settings for the Web Application Extension.

To modify the settings, press the Internet Options button on the Procedure Properties window.

Window Tab

Skeletons are a collection of HTML files that contain all the information needed to control the construction of the delivered HTML pages. These files consist of true HTML code along with the TSSCRIPT scripting language.



The Window tab allows you to override the global skeleton settings for this procedure only. The change will not affect any other procedure.

Override Global settings
Check this box to override the Window settings in the global Web Application Extension template. Checking this box enables the prompts below.

Skeleton to use
Specifies the default window skeleton to use. This is normally a modified version of WINDOW.HTM skeleton from the supplied skeleton files.

Theme
Skeleton files can be categorized into common themes or styles so all window representations in a theme have a common look and feel. Specify the default winodw theme here.

Extra capabilities
Specifies extra capabilities of one skeleton versus another. The capability is specified here. This is referring to a TSSCRIPT property. This is covered in more detail in Chapter Six of this guide.

Return if launched from browser
Check this box to disable the procedure when the application is run over the Web. This allows you to remove functionality for the Web version of your application without removing it from the Windows version.

Report Tab

The Report tab defines how the report title and page number will display on the generated HTML page. By default all internet reports will contain a toolbar at the top of the generated HTML page. This toolbar give the following functionality:

First Page, Previous Page, Next Page, Last Page, Zoom In, Zoom Out, One Page, Two Pages, and Exit.

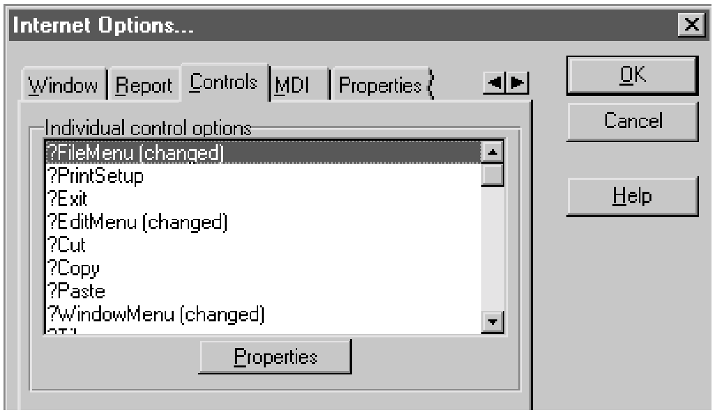
The reports tab contains the following template prompts:

Previewer Window Title
Specifies the title of the report should display in the report preview window. This title will display in the internet explorer window title as well as at the top of the HTML page above the report. This must be a string.

Include current page in title?
Check this box to display the current page number in the report previewer window as well as at the top of the HTML page above the report.

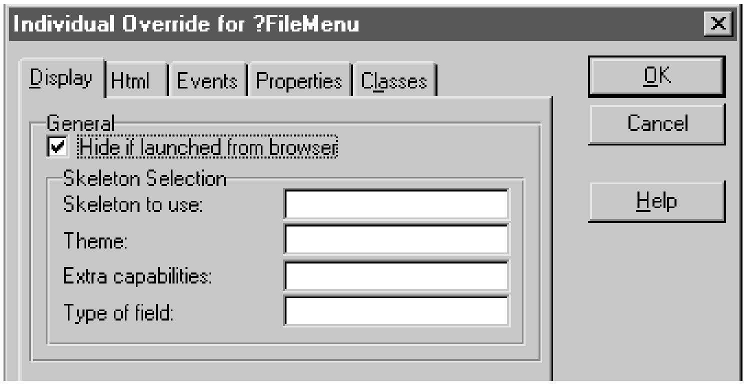
Show total Pages in title?
Check this box to display the total number of pages in the report previewer window (next to the current page number) as well as at the top of the HTML page above the report (next to the current page number).

Controls Tab



Individual Control Options
Hightlight a control in the listbox and press the Properties button to modify specific control options.

Display



☑ Hide if launched from browser

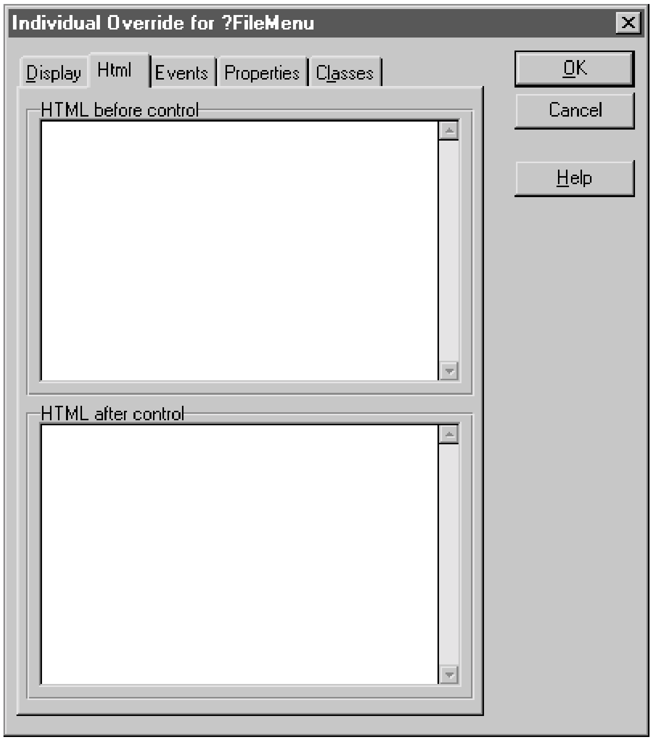
Skeleton to use
Specifies the default window skeleton to use. This is normally a modified version of WINDOW.HTM skeleton from the supplied skeleton files.

Theme
Skeleton files can be categorized into common themes or styles so all window representations in a theme have a common look and feel. Specify the default winodw theme here.

Extra capabilities
Specifies extra capabilities of one skeleton versus another. The capability is specified here.

Type of field
This is for fields which need special formatting such as dates, times, and monetary pictures.

HTML

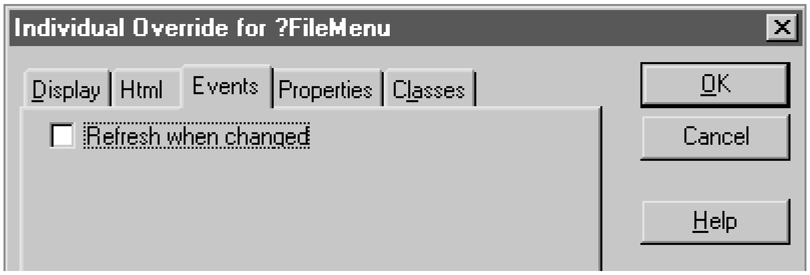


One of the most powerful features of the WBC Templates is the ability to embed HTML code in the HTML pages which are output by the Web-enabled application. This feature allows you to add any HTML code at points before or after any control on the resulting Web page. This code does not affect the application when it is running as a Windows executable.

Using Embedded HTML, you can write any HTML code supported by the browser. You can insert your own custom JavaScript, Java applets, ActiveX controls, Shockwave files, or other objects.

Events

This tab allows you to override the page submission event for a control.



☐ Refresh when changed

Refresh when changed

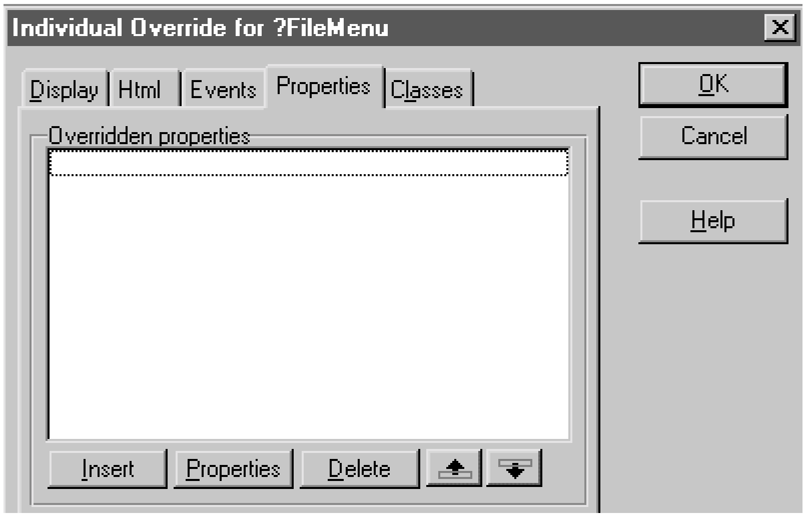
Check this box to cause the page to be submitted to the server when the value of the control changes. The press of a command button automatically causes a page submission. Most other controls that allow data entry do not automatically submit the page to the browser.

This means the processing of events associated with the control is delayed until the page is submitted to the browser. Your embedded code would not execute at the expected time (e.g., code in the Event:Accepted embed point for a control would not execute until the OK button submitted the page). This option allows you to override the page submission event.

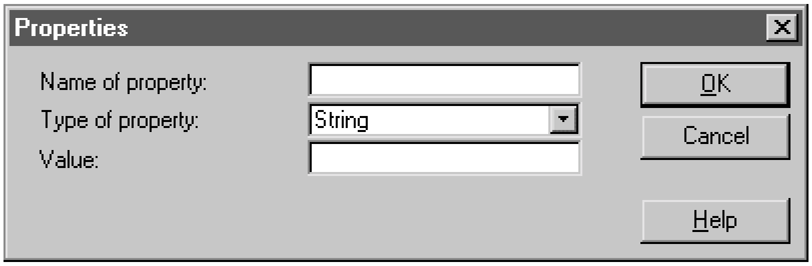
The ability to override the default page submission event when the application is executed in a browser allows you to optimize the application for the Web environment and ensure that all of your embedded code is executed at the time you expect it to.

Properties

A Property is a predefined or customized attribute that is defined in a skeleton file. Through this dialog the skeleton's property can be accessed and executed. Properties serve as a way to translate information about a window or control from the executable to the dynamically generated HTML page.



Press the **Insert**, **Properties** or **Delete** button to modify the properties that the application will look for in the skeleton files.



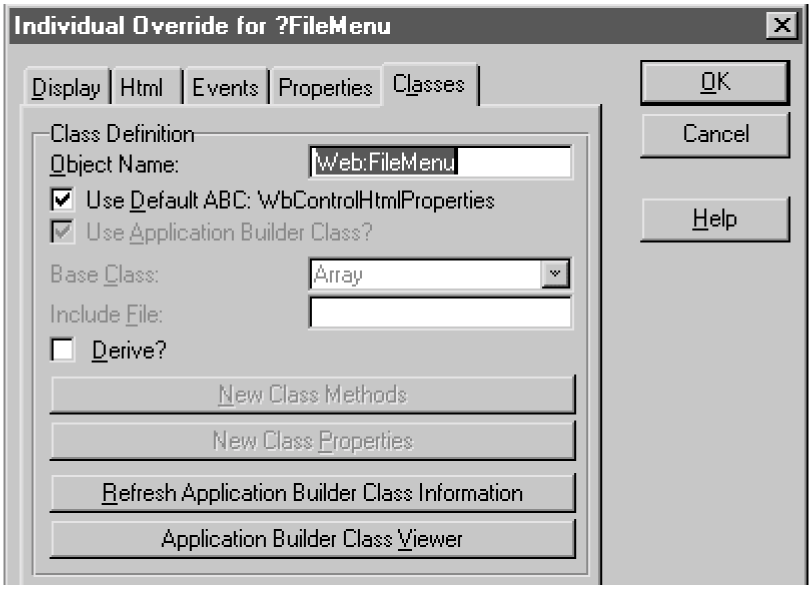
Name of Property
Enter the name of the TSSCRIPT property defined in the skeleton file.

Type of Property
Select the data type from the dropdown list. Select from BOOL, Integer, String, or Reference.

Value
Enter a literal value or a valid clarion language expression.

Classes

The Classes Tab lets you specify which classes (objects) the templates use to accomplish various tasks, and the source modules that contain the class definitions. This approach gives you the capability to use as much of the WBC Library as you want and as much of your own classes as you want.


Individual Override for ?FileMenu
Display
Html
Events
Properties
Classes
Class Definition
New Class Methods
New Class Properties
Refresh Application Builder Class Information
Application Builder Class Viewer
OK
Cancel
Help

Object Name: Web:FileMenu

☑ Use Default ABC: WbControlHtmlProperties

☑ Use Application Builder Class?

Base Class: Array

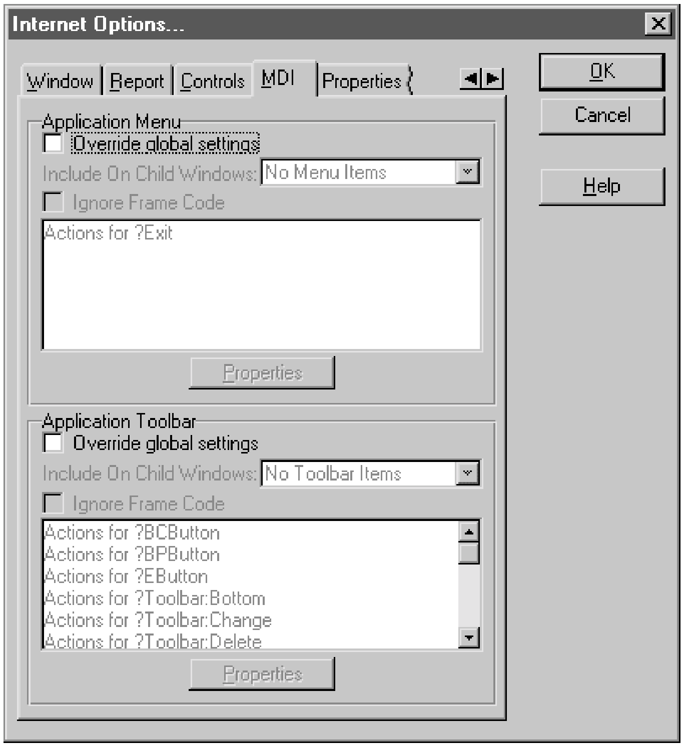
Include File:

☐ Derive?

To change the class for an item or override the class, highlight it in the list, then press the **Properties** button.

MDI Tab

This section determines the manner in which Application Menus and Toolbars are handled.



Application Menu

☐ Override global settings

Include On Child Windows: No Menu Items

☐ Ignore Frame Code

Actions for ?Exit

Application Toolbar

☐ Override global settings

Include On Child Windows: No Toolbar Items

☐ Ignore Frame Code

Actions for ?BCButton
Actions for ?BPButton
Actions for ?EButton
Actions for ?Toolbar:Bottom
Actions for ?Toolbar:Change
Actions for ?Toolbar:Delete

Tip

For control over specific Menu or Toolbar items, set the MDI overrides in the Frame Procedure’s Internet Options.

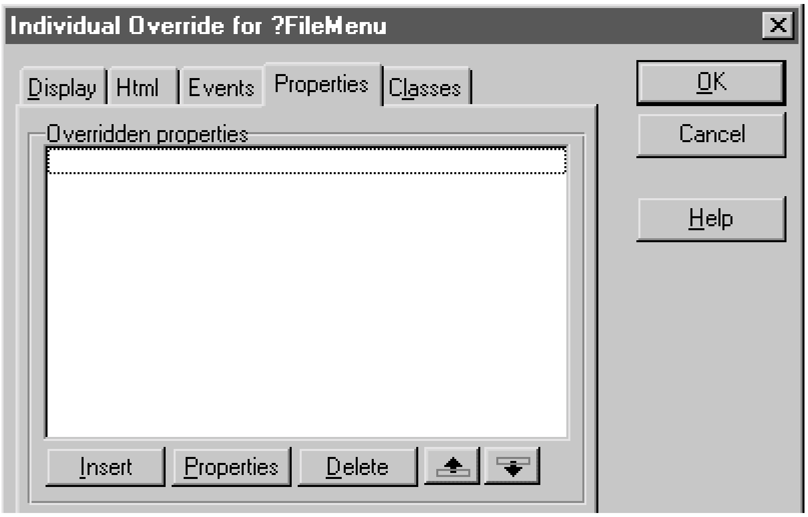
Merge Frame Menu
Check this box to Merge the Frame’s Menu when running this procedure.

Merge Frame Toolbar
Check this box to Merge the Frame’s Toolbar when running this procedure.

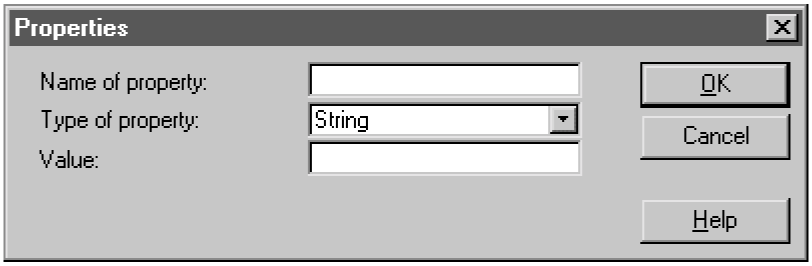
For a Frame Procedure, you have additional options. See *Frame Procedure MDI Options*.

Properties Tab

A Property is a predefined or customized attribute that is defined in a skeleton file. Through this dialog the skeleton's property can be accessed and executed. Properties serve as a way to translate information about a window or control from the executable to the dynamically generated HTML page.



Press the **Insert**, **Properties** or **Delete** button to modify the properties that the application will look for in the skeleton files.



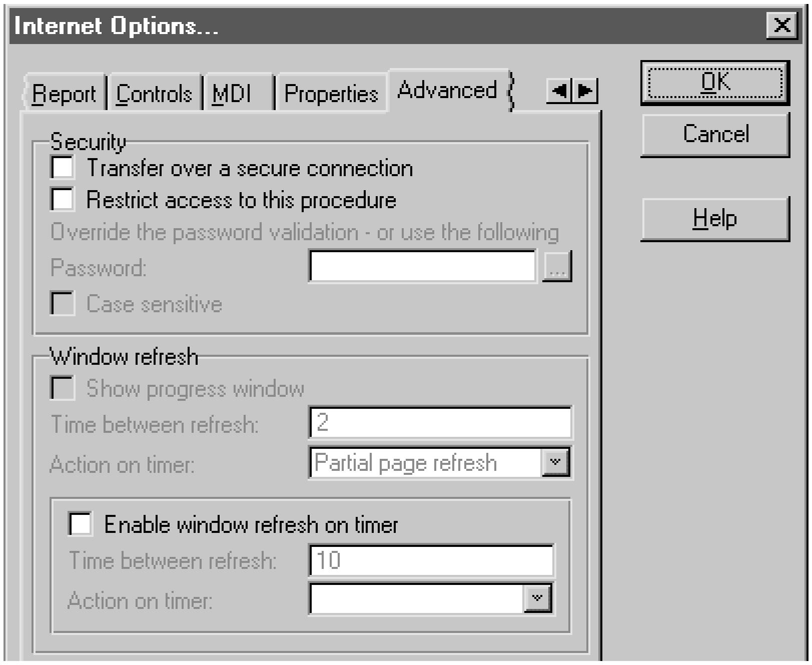
Name of property:
Type of property: String
Value:

Name of Property
Enter the name of the TSSCRIPT property defined in the skeleton file.

Type of Property
Select the data type from the dropdown list. Select from BOOL, Integer, String, or Reference.

Value
Enter a literal value or a valid clarion language expression.

Advanced Tab



Security
☐ Transfer over a secure connection
☐ Restrict access to this procedure
Override the password validation - or use the following
Password:
☐ Case sensitive

Window refresh
☐ Show progress window
Time between refresh: 2
Action on timer: Partial page refresh

☐ Enable window refresh on timer
Time between refresh: 10
Action on timer:

Security

Restrict Access to this procedure
Check this box to password protect the procedure and enable the two fields below.

Password
Specify a password or select a variable from the file schematic by pressing the ellipsis (...) button. A static password provides simple protection.

Case Sensitive
Check this box to enforce case sensitive validation of the password. If the box is not checked, case is ignored.

Window refresh

Show progress window
This controls the window associated with a Report or Process procedure. It is not available for other procedure types. Check this box to display the window associated with the Report Procedure when running over the Web. If not checked, the window is ignored. If the window in a Report Procedure contains a Pause Button control template, the box is checked and cannot be changed. In a Process procedure, the box is checked and cannot be changed. This makes sure the window displays.

Time between refresh
Specify the number of seconds between each refresh.

Action on Timer
Specify the action to perform when the timer event is reached. The choices are:

> *Partial Page refresh*
> Redisplays Java controls and HTML entry controls to reflect current data.
>
> *Submit page*
> Sends data to server application and redraws page as instructed by the server application
>
> *Complete Page refresh*
> Redraws the entire page.

Enable Refresh on timer
Check this box to refresh the entire page or only the page data based on a timer. A TIMER attribute on a WINDOW is independant of this setting. This setting is used on the Web and the TIMER attribute is used when the application runs under Windows.

Tip

This feature should be used sparingly to ensure minimal network traffic.

Time between refresh
Specify the number of seconds between each refresh.

Action on Timer
Specify the action to perform when the timer event is reached. The choices are:

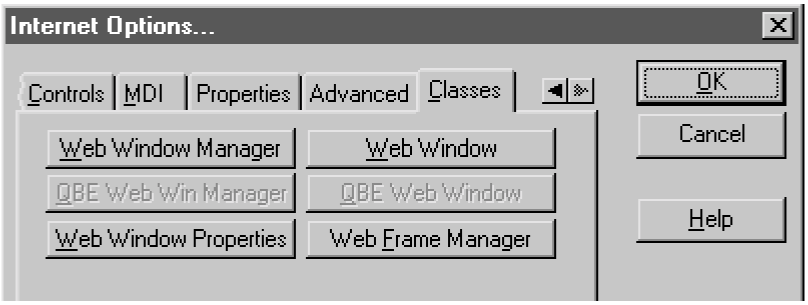
Partial Page refresh
Redisplays Java controls and HTML entry controls to reflect current data.

Submit page
Sends data to server application and redraws page as instructed by the server application

Complete Page refresh
Redraws the entire page.

Classes Tab

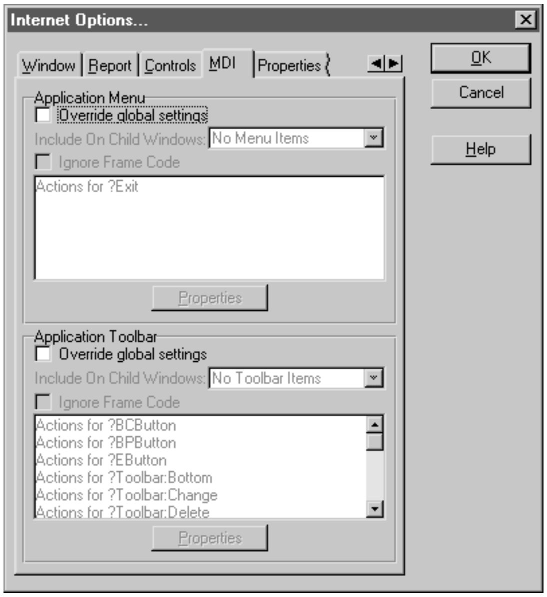
The Classes Tab lets you specify which classes (objects) the templates use to accomplish various tasks, and the source modules that contain the class definitions. This approach gives you the capability to use as much of the WBC Library as you want and as much of your own classes as you want.



To change the class for an item or override the class, highlight it in the list, then press the **Properties** button.

Frame Procedure MDI Options

Application Menu



Include On Child Windows: No Menu Items

Actions for ?Exit

Include On Child Windows: No Toolbar Items

Actions for ?BCButton
Actions for ?BPButton
Actions for ?EButton
Actions for ?Toolbar:Bottom
Actions for ?Toolbar:Change
Actions for ?Toolbar:Delete

Override Global settings
Check this box to override the Menu MDI settings in the global Web Application Extension template. Checking this box enables the other prompts.

Include on Child Windows
Select the option from the drop-down list. The choices are:

Global Setting
Menu choices appear on child windows as specified in the Global options.

All Menu Items
All menu choices appear on child windows.

No Menu Items
No menu choices appear on child windows.

Selected Menu Items
Allows you to select individual menu options from the list below.

Ignore frame code
Check this box to ignore any embedded code in the Application Frame’s ACCEPT loop for menu items.

Application Toolbar

This section determines the manner in which Application Toolbar controls are handled. This allows you to specify which global Toolbar controls are displayed on “child” windows.

Override Global settings
Check this box to override the Toolbar MDI settings in the global Web Application Extension template. Checking this box enables the other prompts.

Include on Child Windows
Select the option from the drop-down list. The choices are:

Global Setting
Toolbar controls appear on child windows as specified in the Global options.

All Toolbar Items
All Toolbar items appear on child windows.

Standard Toolbar Only
Only the Standard Toolbar items appear on child windows.

No Toolbar Items
No Toolbar items appear on child windows.

Selected Toolbar Items
Allows you to select individual Toolbar items from the list below.

Ignore frame code
Check this box to ignore any embedded code in the Application Frame’s ACCEPT loop for toolbar items.

Code Templates

Dynamic HTML Code Template

This code template allows you to insert dynamic HTML code in any of the Internet embed points. This template is only available for Embed points that can write to the delivered HTML page at runtime.

You can specify any valid Clarion expression in the entry box. Any variables used in the expression will use the current value at the time the HTML code is written.

Note:

When creating your expression to write HTML code, you must handle special characters, such as <, by using two characters in succession.

This template uses the Target.WriteLn method to write the value of the expression to the delivered HTML page.

See also: *Embedding HTML*

Static HTML Code Template

This code template allows you to insert static HTML code in any of the Internet embed points. This template is only available for Embed points that can write to the delivered HTML page at runtime.

You can specify any valid HTML code in the entry box.

This template uses the Target.WriteLn method to write the HTML code to the delivered HTML page.

Note:

If you use the Static HTML Code Template, special characters are handled automatically.

GetCookie Code Template

This template allows you to retrieve a cookie from the client’s machine. The following template prompts are provided:

Cookie Name
Provide a name for the cookie. This is the name used in the SetCookie Code template to write the cookie. If the cookie does not exist, a null value is assigned to the Variable to Set.

Variable to Set
Select a variable from the file schematic by pressing the ellipsis (...) button. The value of the cookie is assigned to the variable.

See also: *SetCookie Code Template*, *Cookies (Persistent Client Data)*

SetCookie Code Template

This template allows you to set a cookie on the client’s machine for later retrieval. The following template prompts are provided:

Cookie Name
Provide a name for the cookie. This is the name to use in the GetCookie Code template to retrieve the cookie. If a cookie of the same name exists, it is overwritten.

New Value
Specify a value or select a variable from the file schematic by pressing the ellipsis (...) button. This value is assigned to the cookie.

See also: *GetCookie Code Template*, *Cookies (Persistent Client Data)*

Cookies (Persistent Client Data)

Cookies are a method for Web servers to both store and retrieve information on the client side of the connection. This allows a server to store data on the client's machine and retrieve it later.

A server can send a piece of data to the client (browser) which the client stores locally. This is known as a cookie (the name has no known origin). Cookies contain a range of URLs for which it is valid.

Later, when the client returns to a URL within that range, the server can query the cookie and use that data. A server cannot retrieve information from other servers (i.e., a server cannot query a cookie that is out of its domain range).

This mechanism is similar to the INI file storage and retrieval paradigm in Windows (GETINI and PUTINI) and provides a method for identifying user preferences, and other data.

For example, an application that requires a user to provide their name before entering can use a cookie to avoid the Login process after the first visit.

Note:

Cookies are machine specific so a client who accesses a site from more than one machine will need to provide the cookie information once for each machine so a cookie is stored on the machine. In addition, cookies are browser specific, so a client who uses more than one browser, will need to set and get cookies for each browser.

Your Web-enabled applications can use cookies to store user preferences such as the default city and state for new records. These settings can be retrieved the next time the user runs the application over the Web.

See also: *GetCookie Code Template*, *SetCookie Code Template*

AddServerProperty Code Template

This template allows you to set the value of the specified outgoing http item in the HTTP header. The following prompts are provided:

Property Name
Provide the property name to set.

Property Value
Select a variable from the file schematic by pressing the ellipsis (...) button. The value of the variable is assigned to the property.

See Also : *GetServerProperty Code Template*

GetServerProperty Code Template

This template allows you to get the value of the specified http item in the HTTP header. The following prompts are provided.

Property Name

Provide a name for the HTTP property. If the HTTP field does not exist, a null value is assigned to the Variable to Set.

Variable to Set

Select a variable from the file schematic by pressing the ellipsis (...) button. The value of the property is assigned to the variable.

See Also : *SetServerProperty Code Template*

RedirectToPage Code Template

This template redirects the browser to the designated URL. At the present time, the running program is left running. The program must be terminated or left to time out.

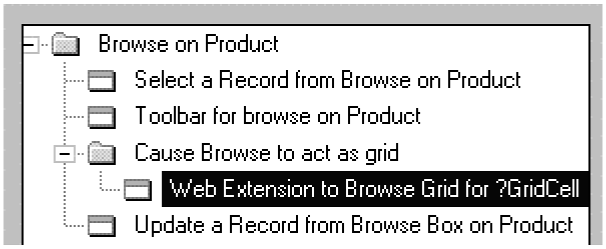
The following prompt is provided:

Page to redirect to

Specifies the URL of the page the browser it redirected to. An absolute or relative URL may be specified.

WebGridExtension

All BrowseBoxes that use the ABC's BrowseGrid control template must use this template for Web applications.



- Browse on Product
  - Select a Record from Browse on Product
  - Toolbar for browse on Product
  - Cause Browse to act as grid
    - Web Extension to Browse Grid for ?GridCell
  - Update a Record from Browse Box on Product

This template requires the use of the ABC BrowseGrid extension.

1. RIGHT-CLICK on the procedure and choose **Extensions** from the popup menu.
2. Within the list of existing extensions, highlight *Cause Browse to act as grid*.
3. Press the INSERT button and select the *WebGridExtension* extension.

This template has no prompts.

For further information about using the browse grid interface, see the BrowseGrid template documentation which can be found in the Template By Topic PDF.

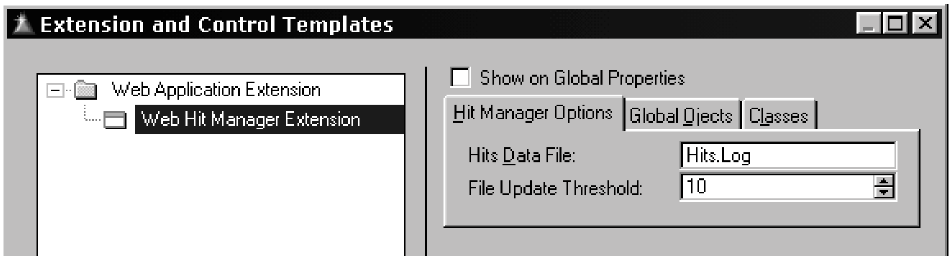
WebHitManager

The WebHitManager extension template provides the ability to record the number of accesses (hits) to an application or certain procedures within the application. Hit counts can be tied to a particular window activity or control event. This extension template is added to the global extension of the application file and allows the Hit Managers global options to be set.

This extension template requires the WebApplicationExtension.

Populating the Template

1. Press the **Global** button from the IDE.
2. Press the **Extensions** button.
3. Highlight the *WebApplicationExtension* template.
4. Press the INSERT button and select the *WebHitManager* extension.


Extension and Control Templates
Web Application Extension
Web Hit Manager Extension
Hit Manager Options
Global Ojects
Classes

☐ Show on Global Properties

Hits Data File: Hits.Log

File Update Threshold: 10

The WebHitManager template provides the following prompts:

Hit Manager Options

Hits Data File
Specify the log file that is used to store the WebHit counts. If no path is specified, the file is created in the Windows directory.

File Update Threshold
Specifies the number of hits to occur before the counts are written to the Hits Data File. The counts are written when the program is terminated, regardless of the threshold specified.

Global Objects

The Global Objects tab lets you specify the default object names for the objects used by the ABC Templates. You can also specify the default classes to be used for the global objects.

Classes

The classes tab lets you control the class (and object) the template uses. You may accept the default Application Builder Class and it's object (recommended) or you may specify your own or a third party class. Deriving your own class can give you very fine control over the procedure when the standard Application Builder Class is not precisely what you need.

See *Template Overview – Classes Tab Options* – Local for complete information on these options.

WebHitProc

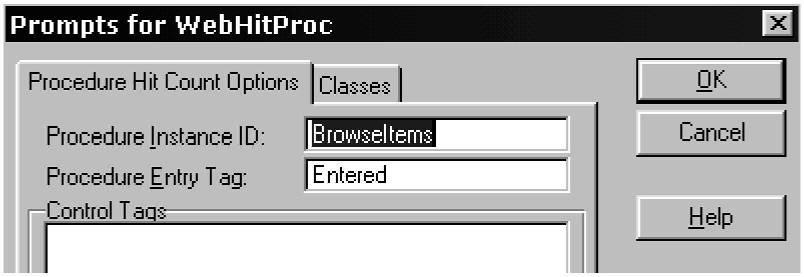
The WebHitProc extension template provides the ability to record the number of accesses (hits) to an application or certain procedures within the application. Hit counts can be tied to a particular window activity or control event. This extension template is added to any procedure that will record the hits tied to the procedure at the procedure entry level, window event level or control event level. This extension requires the application to have the global WebHitManager extension.

This extension template requires the WebHitManger global extension.

Populating the Template

1. RIGHT-CLICK on the procedure and choose **Extensions** from the popup menu.
2. Press the INSERT button and select the *WebHitProc* extension.

Template Prompts


Prompts for WebHitProc

Procedure Instance ID: BrowseItems

Procedure Entry Tag: Entered

Control Tags

The WebHitProc template provides the following prompts:

Procedure Hit Count Options

Procedure Instance Id
Specifies the id of the hit instance recorded in the log file. This id is used to read and write the hit count to the log file. By default this is the name of the procedure.

Procedure Entry Tag
Specifies text to describe the procedure action that is counted. This text is written to the log file. By default this is set to Entered.

Control Tags
Provides a listbox in order to define one or more control specific hit counts.

Link Tag
Specifies text to describe the window or control event that is counted. This text is written to the log file following the Procedure Instance Id.

Trigger Control
Select a control from the drop down listbox. This will trigger the hit count to be incremented when the control is handled and the Trigger Event occurs. To trigger a hit count on a window event, leave the Trigger Control blank.

Trigger Event
Select an event from the drop down listbox. This will trigger the hit count to be incremented when the event occurs. If a control is specified as the Trigger Control, the event is based on the control. If no control is specified, the event is based on the window.

Classes

The classes tab lets you control the class (and object) the template uses. You may accept the default Application Builder Class and it's object (recommended) or you may specify your own or a third party class. Deriving your own class can give you very fine control over the procedure when the standard Application Builder Class is not precisely what you need.

See Template Overview – Classes Tab Options – Local for complete information on these options.

WebShowHits

The WebShowHits extension provides the ability to display a hit count. This extension requires the application to have the WebHitProc extension.

This extension template requires the WebHitProc extension.

Populating the Template

1. RIGHT-CLICK on the procedure and choose **Extensions** from the popup menu.

2. Press the INSERT button and select the *WebShowHits* extension.

Template Prompts

The WebShowHits template provides the following prompts:

Count Tag
Specifies the tag defined in the WebHitsProc extension for a Procedure Entry Tag or Control Link Tag. This tag is used to retrieve and display an up to date count. This tag is case sensitive.

Assignment Type
Select Text Property or Variable from the drop down listbox. This assigns the specified control or variable the value of the count for display.

Control to Receive Link
Specify the window control that will display the hit count. This is enabled only when Text Property is selected as the Assignment Type.

Variable to Receive Link
Specify the variable that will display the hit count. This is enabled only when Variable is selected as the Assignment Type.

WebGuard Application Extension

The WebGuard Application extension template provides an easy way to limit access to applications at the application and/or procedure level. The template provides a default logon window (this window may be overridden) used for logging in to an application and provides the ability to add a new user to the application.

WebGuard supports the ability to define specific capabilities (rights) to each user. These capabilities are used by the WebGuardProc procedure extension template to validate the users rights to the requested information. WebGuard has the ability to work in conjunction with the GlobalDocumentHandling extension (Internet Toolkit). By combining these templates the ability to email a user about invalid logins to their account is enabled.

The minimum requirement of keys and columns needed to use the WebGuard extension is defined below. The column names can be anything; they do not have to match the definition. The purpose of the keys and columns are the important issue.

Required:

CUSTOMER	**FILE**
NameKey	**KEY, Unique, Contains Name column**
Name	**FIELD, must be a STRING, CSTRING, or PSTRING**
Password	**FIELD, must be a STRING, CSTRING, or PSTRING**
Capability	**FIELD, must be a STRING, CSTRING, or PSTRING**
Number	**FIELD**

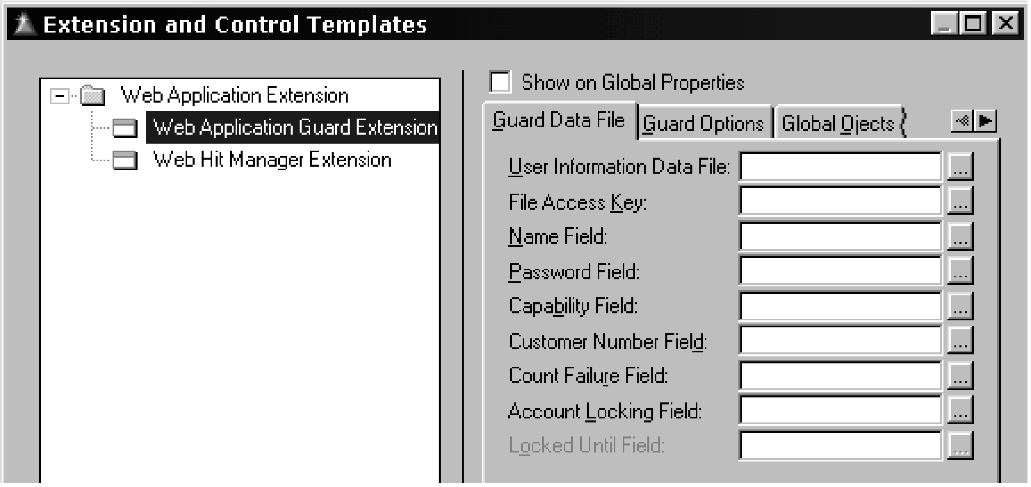
Optional: The following fields are required only when enabling their corresponding options.

CountFailure	**FIELD**
DaysToLock	**FIELD**
MaxLogonAttempts	**FIELD**
AccountLocking	**FIELD**
LockedUntil	**FIELD**

This extension template requires the global WebBuilder template.

Populating the Template

1. Press the Global icon button from the IDE.
2. Press the **Extensions** button.
3. Highlight the *WebApplicationExtension* extension template.
4. Press the INSERT button and select the *WebGuard* extension.



The WebGuard template provides the following prompts:

Guard Data File

User Information Data File
Select the file to be used as the customer (Customer) file. Use the ellipsis (…) to select the file from the file schematic or type in a file that exists in the file schematic.

File Access Key
Select the key that is made up of the customer name field. This key is used to retrieve a specific customer record from the Customer file. The key should be a unique key. Use the ellipsis (…) to select the key from the customer file or type in a key that exists in the file.

Name Field
Select the field to be used as the customer name field. This field specifies the customer name and must be defined as a STRING, CSTRING, or PSTRING. The customer name field must be the primary field in the *File Access Key*. Use the ellipsis (…) to select the field from the customer file or type in a field that exists in the file.

Password Field
Select the field to be used as the customer password field. This field will contain the customer's defined password and must be defined as a STRING, CSTRING, or PSTRING. Use the ellipsis (…) to select the field from the customer file or type in a field that exists in the file.

Capability Field
Select the field to be used to validate customer capabilities. This field must be defined as a STRING, CSTRING, or PSTRING. Use the ellipsis (…) to select the field from the customer file or type in a field that exists in the file. Capabilities define the specific abilities available to a customer. For example, there may be several types of customers that have different rights in the system. There may be a PRIORITY customer and a STANDARD customer. If a PRIORITY customer logs into the system they will potentially see different menu choices than the STANDARD customer.

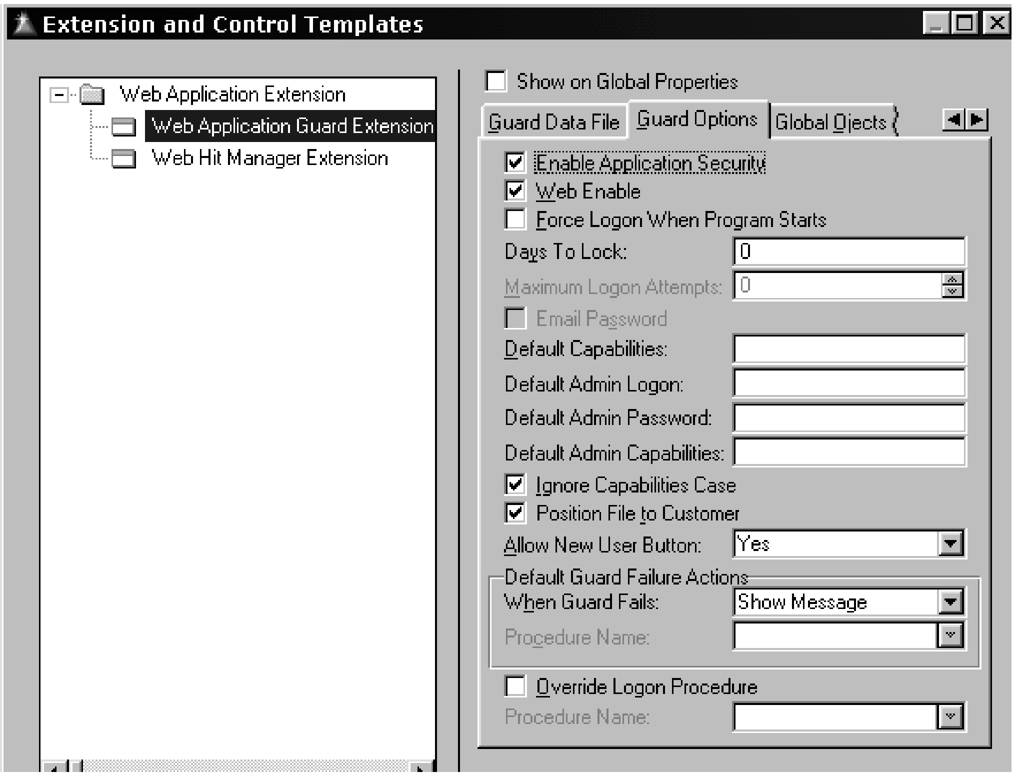
Customer Number Field
Select the field to be used as the customer number field. This field will contain the customer id used to identify a customer. Use the ellipsis (…) to select the field from the customer file or type in a field that exists in the file. This field should exist as part of a autoincrementing key so new users will have incremented customer numbers.

Count Failure Field
Select the field to be used as a count field. This field is incremented when an invalid logon occurs. When the invalid count exceeds the *Maximum Logon Attempts*, the customer account can be locked either for the specified number of days or until a specified date. Use the ellipsis (…) to select the field from the customer file or type in a field that exists in the file.

Account Locking Field
Select the field to be used as the lock status field. This field is set when the Maximum Logon Attempts is reached. Use the ellipsis (…) to select the field from the customer file or type in a field that exists in the file.

Locked Until Field
Select the field to be used to specify the date the customer account will be unlocked. This field is set when the Maximum Logon Attempts is reached. It is set to the current date plus the number specified in the Days To Lock template prompt. Use the ellipsis (…) to select the field from the customer file or type in a field that exists in the file.

Guard Options


Extension and Control Templates
Web Application Extension
Web Application Guard Extension
Web Hit Manager Extension
Guard Data File
Guard Options
Global Ojects

☐ Show on Global Properties

☑ Enable Application Securit
☑ Web Enable
☐ Force Logon When Program Starts
Days To Lock: 0
Maximum Logon Attempts: 0
☐ Email Password
Default Capabilities:
Default Admin Logon:
Default Admin Password:
Default Admin Capabilities:
☑ Ignore Capabilities Case
☑ Position File to Customer
Allow New User Button: Yes

Default Guard Failure Actions
When Guard Fails: Show Message
Procedure Name:

☐ Override Logon Procedure
Procedure Name:

Enable Application Security
Check this box to enable the WebGuard application extension for an application. By default this box is checked. When this box is unchecked, all WebGuard prompts are disabled.

Web Enable
Check this box to enable WebGuard login windows to work in a web application. By default this box is checked.

Force Logon When Program Starts
Check this box to have a logon window appear at the start of the application. By default this box is unchecked.

Days To Lock
Specify the number of days a customer account will be locked in the case when the *Maximum Logon Attempts* occur.

Maximum Logon Attempts
Specify the maximum number of invalid logon attempts. This is available when a Count Failure field is specified. *Days To Lock* and *Account Locking Field* must be entered in order for the Maximum Logon Attempts to be validated.

Email Password
Check the box to have an email sent to the customer when the maximum number of invalid logon attempts occur. See the *Global Document Handling* Internet ToolKit extension to setup email specifications.

Default Capabilities
Specifies a string, variable, or runtime expression using EVALUATE to use as the default capability settings for all customers logging in to the system. To specify a variable here, precede the entry with an exclamation point (!). To specify a runtime expression, precede the entry with an equal sign (=).

Default Admin Logon
Specifies a string, variable, or runtime expression using EVALUATE to use as the default Administrator logon name. To specify a variable here, precede the entry with an exclamation point (!). To specify a runtime expression, precede the entry with an equal sign (=).

Default Admin Password
Specifies a string, variable, or runtime expression using EVALUATE to use as the default Administrator password. To specify a variable here, precede the entry with an exclamation point (!). To specify a runtime expression, precede the entry with an equal sign (=).

Default Admin Capabilities
Specifies a string, variable, or runtime expression using EVALUATE to use as the default Administrator capabilites. To specify a variable here, precede the entry with an exclamation point (!). To specify a runtime expression, precede the entry with an equal sign (=).

Ignore Capabilities Case
Check this box to force the capabilities verification to be case insensitive. The default value for this prompt is case insensitive.

Position File to Customer
This is not implemented at this time.

Allow New User Button
Select Yes, No, or Use External Procedure to have the New User button shown on the window or not.

Default Guard Failure Actions
Define the failure actions to take when a customer does not have the required capabilities to enter a specific area. These default options can be overridden using the WebGuard Procedure extenson.

When WebGuard Fails
Choose *Show Message* or *Run a Procedure* as the default failure action. Show message displays a message to the user to inform them about their capabilites. The message is defined in WebGuard.trn.

Procedure Name
Choose an existing procedure from the drop down listbox or type in a new procedure name. This procedure is executed when a user tries to enter a procedure with invalid capabilities.

Override Logon Procedure
Check this box to override the default logon window in order to provide a customized one.

Procedure Name
Choose an existing procedure from the drop down listbox or type in a new procedure name. This procedure is used to replace the default logon window and code. This procedure must return a return value of a BYTE. The return field returns the error severity. The severity levels can be found in ABERROR.INC.

Global Objects

The Global Objects tab lets you specify the default object names for the objects used by the ABC Templates. You can also specify the default classes to be used for the global objects.

Classes

The Classes tab lets you control the classes (and objects) the procedure uses. You may accept the default Application Builder Class and its object (recommended), or you may specify your own or a third party class. Deriving your own class can give you very fine control over the procedure when the standard Application Builder Class is not precisely what you need. See Template Overview—Classes Tab Options—Local for complete information on these options.

WebGuardProc Procedure Extenstion

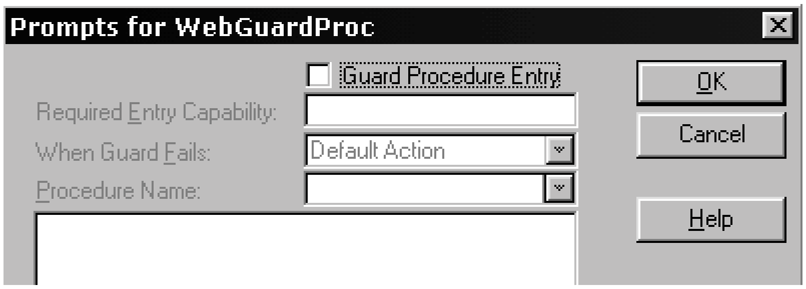
The WebGuardProc procedure extension is available when the WebGuard application extension is added globally to the application. This procedure extension gives the ability to limit access to specific procedure based on defined user capabilities.

This extension template requires the global WebGuard extension.

Populating the Template

1. RIGHT-CLICK on the procedure and choose **Extensions** from the popup menu.
2. Press the INSERT button and select the *WebGuardProc* extension.

Template Prompts



☐ Guard Procedure Entry

Required Entry Capability:

When Guard Fails: Default Action

Procedure Name:

The WebGuardProc extension provides the following prompts:

Guard Procedure Entry
Check this box to validate the users capabilities (rights) to access the procedure.

Required Entry Capability
Specify the required capability to gain access to the procedure.

When Guard Fails
Choose *Default Action*, *Show Message* or *Run a Procedure* as the failure action. The failure actions define the action to take when a customer does not have the required capabilities to enter a specific area. Show message displays a message to the user to inform them about their capabilites. The message is defined in WebGuard.trn.

Procedure Name
Choose an existing procedure from the drop down listbox or type in a new procedure name. This procedure is executed when a user tries to enter a procedure with invalid capabilities.

Control To Guard
Specific controls can be guarded by this extension. Choose the control to guard from the drop down listbox.

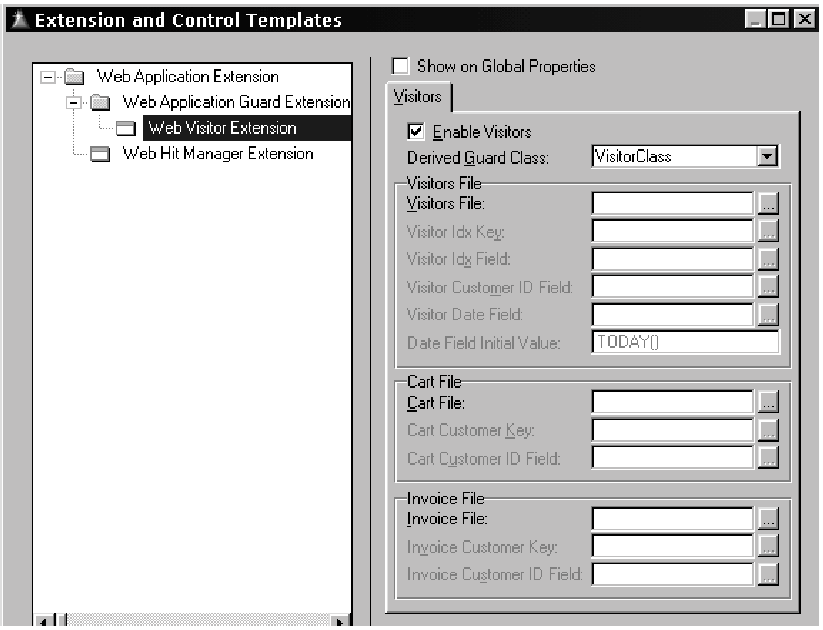
Required Control Capability
Specify the required capability to gain access to the control.

Guard Type
A control can either be hidden or trigger a failure action if the capabilities requirement is not met. Select *Hide* or *Trigger* from the drop down listbox.

When Guard Fails
Choose *Default Action*, *Show Message* or *Run a Procedure* as the triggered failure action. The failure actions define the action to take when a customer does not have the required capabilities for the specified control. Show message displays a message to the user to inform them about their capabilites. The message is defined in WebGuard.trn.

Procedure Name
Choose an existing procedure from the drop down listbox or type in a new procedure name. This procedure is executed when a user tries access a control with invalid capabilities.

WebVisitor



☐ Show on Global Properties

☑ Enable Visitors

Derived Guard Class: VisitorClass

Date Field Initial Value: TODAY()

The WebVisitor extension template is a global extension that allows an application to have temporary users (visitors). This concept is most often used in a shopping cart application where users can view products and use the shopping cart prior to signing in and processing an order. The template creates a temporary user record in the visitor file as well as the customer file. This template requires the WebGuard global extension.

Enable Visitors
Check this box to enable Visitors for the application.

Derived Guard Class
Select the class the template will use from the drop down listbox. You may accept the default VisitorClass (reccomended) or you may specify your own or a third party class. Deriving your own class can give you very fine control over the functionality when the standard Application Builder Class is not precisely what you need.

Visitors File
Select the file to be used as the visitor (Visitor) file. Use the ellipsis (…) to select the file from the file schematic or type in a file that exists in the file schematic.

Visitor Idx Key
Select the key to be uses as the Visitor Idx key. This key should be an auto-incrementing key that consists of the visitor idx field. Use the ellipsis (…) to select the field from the visitor file or type in a field that exists in the file.

Visitor Idx Field
Select the field to be used as the visitor idx field. Use the ellipsis (…) to select the field from the visitor file or type in a field that exists in the file.

Visitor Customer Id Field
Select the field to be used as the customer id field. This is used to relate the visitor record to the customer file. This is a one-to-one relationship. Use the ellipsis (…) to select the field from the visitor file or type in a field that exists in the file.

Visitor Date Field
Select the field to be used as the visitor date field. This is used to keep track of the date the visitor signed on to the system and is also used by the DeleteVisitorProcess to remove obsolete visitors. Use the ellipsis (…) to select the field from the visitor file or type in a field that exists in the file.

Date Field Initial Value
Specify the initial date value to be used when a record is added to the visitor file. This may be a value, function or variable. The default value is TODAY(). The initial value may also be set in the initial value of the date field in the dictionary.

Cart File
Select the file to be used as the shopping cart (Cart) file. Use the ellipsis (…) to select the file from the file schematic or type in a file that exists in the file schematic.

Cart Customer Key
Select the key to be uses as the customer key. This key should consist of the customer id field. It is used to relate the cart file to the customer file. Use the ellipsis (…) to select the field from the cart file or type in a field that exists in the file.

Cart Customer Id Field
Select the field to be used as the customer id field. This field identifies the id of the customer who created the shopping cart. It is used to relate the Cart file to the Customer file. The relationship is a one-to-many relation. Use the ellipsis (…) to select the field from the shopping cart file or type in a field that exists in the file.

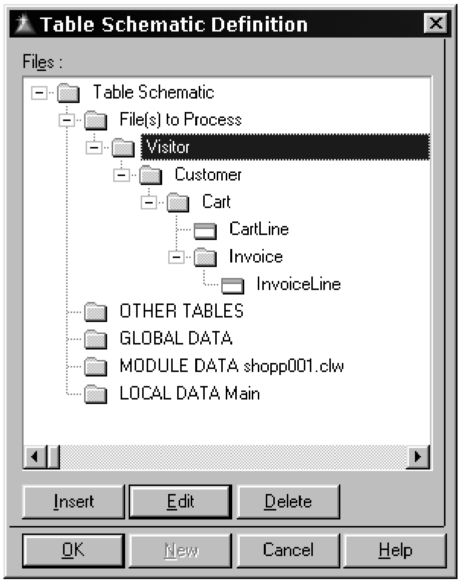
Invoice File
Select the file to be used as the invoice (Invoice) file. Use the ellipsis (…) to select the file from the file schematic or type in a file that exists in the file schematic.

Invoice Customer Key
Select the key to be uses as the customer key. This key should consist of the customer id field. It is used to relate the invoice file to the customer file. Use the ellipsis (…) to select the field from the invoice file or type in a field that exists in the file.

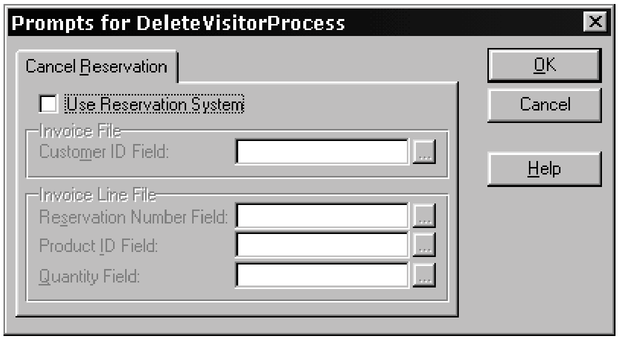
Invoice Customer Id Field
Select the field to be used as the customer id field. This field is used to relate the Invoice to a Customer. Use the ellipsis (…) to select the field from the invoice file or type in a field that exists in the file.

DeleteVisitorProcess

The DeleteVisitorProcess extension template will remove old non-existant visitors from the visitor table and all related tables. This extension may only be added to a PROCESS procedure. The following files must be in the file schematic of the procedure in order for all related tables to be cleaned of the obsolete visitor records.



The PROCESS procedure should have the Actions for Process set to DELETE record. Check the Use RI constraints box. The record filter should be set to filter out visitor records older than x number of days. To delete all visitor records older than 7 days set the record filter to VIS:Dte < TODAY()-7. VIS:Dte is the Visitor Date Field defined above. To delay showing the process window, use the ExtendedProgressWindow extension.



Use Reservation System
Check this box use the reservation system.

Customer ID Field
Select the field to be used as the customer id field. Use the ellipsis (…) to select the field from the invoice file or type in a field that exists in the file.

Reservation Number Field
Select the field to be used as the reservation number field. Use the ellipsis (…) to select the field from the invoice line file or type in a field that exists in the file.

Product ID Field
Select the field to be used as the invoice line product id field. Use the ellipsis (…) to select the field from the product line file or type in a field that exists in the file.

Quantity Field
Select the field to be used as the invoice line quantity field. Use the ellipsis (…) to select the field from the invoice line file or type in a field that exists in the file.

4 - TSSCRIPT

Introduction

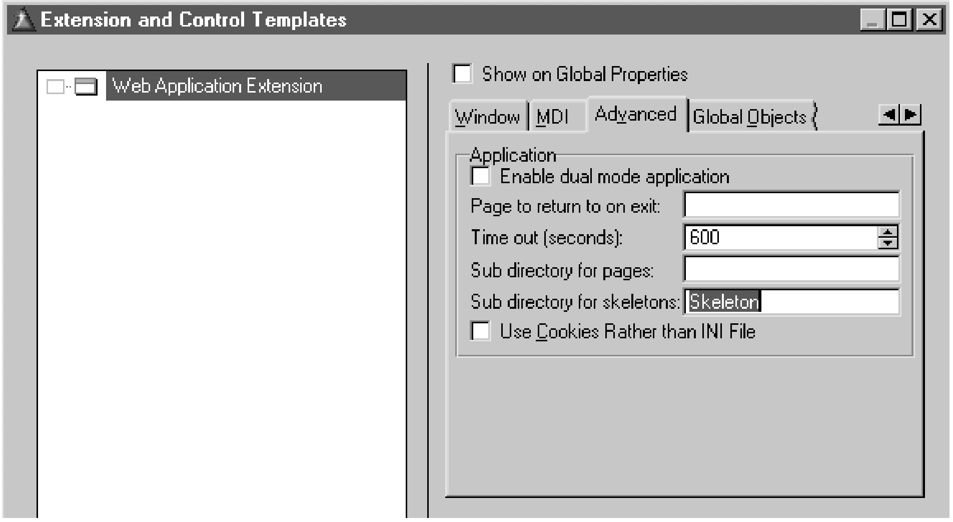
The WebBuilder extensions use a new method of constructing the HTML representation of an application at runtime. You can still embed HTML in your Clarion App as before, but now there are extended capabilities that can be utilized after the app is compiled.

This also provides the ability to change an application's look and feel after the application is made without having to recompile the application. This allows you to easily make your Web application look like your Web site. When you change your Web site's appearance, you can easily change the application's look to match.

The result is an application that controls business logic and data access, and HTML files which control the presentation layer. A non-programmer (i.e., webmaster) can edit the HTML skeletons without the Clarion developer.

Skeletons

Clarion's Web Builder templates use a collection of HTML files called Skeletons. These files contain all the information needed to control construction of the delivered HTML page. These files are stored in the directory named in the Global Extension of your web-enabled app. The current default is *Skeleton*.



Extension and Control Templates

Web Application Extension

☐ Show on Global Properties

Application

☐ Enable dual mode application

Page to return to on exit:

Time out (seconds): 600

Sub directory for pages:

Sub directory for skeletons: Skeleton

☐ Use Cookies Rather than INI File

The collection of files is first read by the web-enabled application when it executes and all of the possible options are stored in an internal database.

When it is time for the app to construct a page, the database is queried and the application uses a skeleton which best matches the control.and its properties. If you examine your Skeleton folder, you will notice files such as button.htm, prompt.htm, string.htm, etc. While the filenames are irrelevant (unless you explicitly specify a WebStyle file to use for a control), examination of the meta tags in the HTML files will show you the properties of the skeleton.

There are four primary properties of a control skeleton which are used to determine the best match at runtime: Control Type, Style, Capabilities, and Field Type.

The first is determined by the WINDOW definition:

Control Type

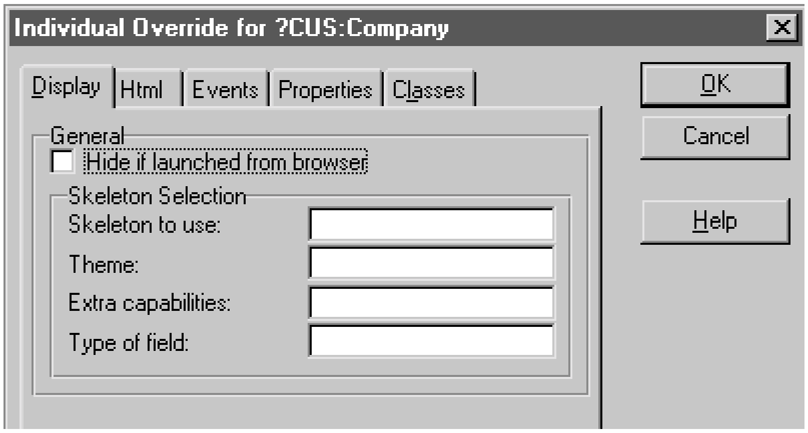
The type of control populated on the WINDOW (e.g., BUTTON, STRING, ENTRY, etc.).

EXAMPLE:

```
<meta name=”ts-control” content=”button”>
```

The other three are determined by values you enter in the Individual Overrides for a control which allows you to specify properties that the app will consider when finding the best match at runtime.

The other three are determined by values you enter in the Individual Overrides for a control which allows you to specify properties that the app will consider when finding the best match at runtime.



Skeleton to Use

This “hard-wires” a specific HTML skeleton to the control. If you specify a filename here, no other properties are considered.

Style

This property allows you to categorize skeletons into a common theme or style. For example, you can replicate all of the standard skeletons and add a “western” style to the new skeletons (e.g., images of cactus, wood grain buttons, etc.).

Extra Capabilities

This property allows you to specify certain capabilities in your skeleton.

Examples:

In the current skeletons, a TAB can be represented in two ways:

with the selected TAB appearing on top and the rest hidden

or

with all TABs showing on a taller page.

This is controlled by specifying the ***showall*** property in the **Capabilities** prompt in the Internet Connect template in the IDE (individual overrides for a control).

In TAB.ALL.HTM, you will find these two meta-tags:

```
<meta name=”ts-control” content=”tab”>
<meta name=”ts-capabilities” content=”showall”>
```

In TAB.ONE.HTM, you only find a meta-tag for the control Type:

```
<meta name=”ts-control” content=”tab”>
```

Therefore, if you specify the ***showall*** capability property in the **Capabilities** prompt in the WebBuilder template in Clarion, it signifies that TAB.ALL.HTM best matches and is the one used.

You can use any words as capability keywords. A complete list of the ones included in the standard skeletons will be published later (after more are utilized). In this release the following are used:

```
list.htm: <meta name=”ts-capabilities” content=”drop”>
```

Supports droplists

```
query.htm: <meta name=”ts-capabilities” content=”query”>
```

Supports the query button control template

```
splash.htm: <meta name=”ts-capabilities” content=”splash”>
```

Supports a splash window which closes after the time specified in the APP

tab.all.htm: <meta name=”ts-capabilities” content=”showall”>

Supports all TABs showing on a taller page.

table.htm: <meta name=”ts-capabilities” content=”multicolumn,pageloaded,default”>

Multi-column Listbox support (as an HTML table).

Type of Field

This property has not yet been utilized by the current set of skeletons, but its intended use is for fields which need special formatting such as dates, times, and monetary pictures.

TSScript

The SoftVelocity scripting language extends the HTML skeleton technology by allowing additional formatting and conditional options in a skeleton file. Although the scripting language is fairly simple in design, it is flexible enough to support complex logic and conditional generation of html from a Clarion application. A few examples are included at the end of this chapter.

Basic Structure

<TSSCRIPT> </TSSCRIPT>

All script code is enclosed in a pair of tags. <TSSCRIPT> begins a block of code and </TSSCRIPT> terminates a block. These can be nested.

Example:

```
<TSSCRIPT tag=a attr=href replace=NAME value=Contents>
<A HREF=”mailto:NAME”>
<TSSCRIPT value=Contents>
</TSSCRIPT>
</A>
</TSSCRIPT>
```

Patching

One purpose of these skeleton files is to allow data to replace certain elements so that it can be delivered in a manner to display in a browser.

`tag=<name>`

which tag to target; defaults to plain text

Example:

```
<TSSCRIPT tag=a attr=href replace=NAME value=Contents>
<A HREF=”http://NAME”>
<TSSCRIPT value=Contents>
</TSSCRIPT>
</A>
</TSSCRIPT>
```

`tag=*`

Specifies any tag

`attr=<name>`

Specifies the tag attribute to target. The attribute is inserted if does not exists.

`replace=<string>`

The search string to replace with the value attribute. The entire search string is replaced.

Example:

```
<TSSCRIPT tag=a attr=href replace=NAME value=Contents>
<A HREF=”http://NAME”>Click Here</A>
</TSSCRIPT>
```

`patch=<wildcard>`

This is the same as replace, but can contain an asterisk (*) as a wildcard. For example, “think * should”. An asterisk can also match to start or end.

`block=<tag>`

This restricts substitutions to within a specified <tag>.

`value=expression`

The computed value to replace with.

`text=string`

This is the literal text to replace. If omitted (i.e., no value) it removes an attribute or tag.

`type=text|value|html`

Repeats

`repeat=count`

Duplicates the following code for the number of times specified (count).

`name=<id>`

Create a local variable <id> which is bound to the count.

Includes

include=<condition>

Includes the matched items if the condition is true.

omit=<condition>

Includes the matched items if the condition is false.

scope=<name>

Selects the control being addressed by the html.

General

when=<condition>

Specifies to only replace if the expression is true.

phase=<phase,phase>

Specifies which phase(s) the tag should be processed in. If not specified, it is processed as soon as the expressions can be evaluated.

`comment=”...”`

Used to comment.

<TSINCLUDE Name=”displayText.htm”>

Inserts another skeleton file at the location.

META Tags

The term *meta* is derived from the Greek word which denotes a nature of a higher order. Meta data typically consists of a number of pre-defined elements representing specific properties of a resource, and each of these elements can have one or more values.

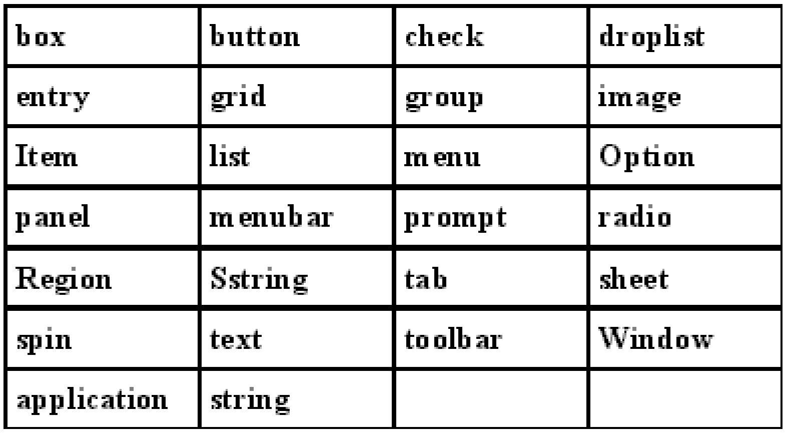
Meta tags were introduced into HTML to allow web authors to specify document properties without displaying them in a browser. The most common use of meta tags is to add keywords and a description to a static web page for search engines. Meta tags can be used to store any document-wide data. For example, you can specify a document’s author, creation date, and last modified date. Some HTML authoring tools automatically add some of these meta data elements.

Clarion uses meta tags to supply properties to skeleton files. This data is later collected at runtime to determine which skeleton to use for a specific control. Meta tags are inserted between the <head> and </head> tags.

The following meta tag names are used in the skeleton files:

`<meta name=”ts-control” content=”controltype,controltype”>`

This tag specifies the control type(s) which the skeleton supports. The possible control types are:



box	button	check	droplist
entry	grid	group	image
Item	list	menu	Option
panel	menubar	prompt	radio
Region	Sstring	tab	sheet
spin	text	toolbar	Window
application	string		

<meta name=”ts-capabilities” content=”capability,capability”>

This tag specifies the capabilities which the skeleton supports.

<meta name=”ts-style” content=”style”>

This tag specifies the style(s) which the skeleton supports.

<meta name=”ts-type” content=”fieldtype,fieldtype”>

This tag specifies the field type(s) which the skeleton supports.

WebStyle Examples

Email String

This skeleton formats data from a variable containing an email address so it is a “Mailto:” hyperlink. To use this skeleton, you would specify the email capability property in the Capabilities prompt in the Internet Connect template in the IDE (individual overrides for a control).

```
<HTML>
<head>
<meta name=”ts-control” content=”sstring”>
<meta name=”ts-capabilities” content=”email”>
</head>
<BODY>
<!— email.string.htm — Start —>
<TSSCRIPT value=”EmbedBeforeControl” type=html>
</TSSCRIPT>
<TSSCRIPT tag=a attr=href replace=NAME value=Contents>
<A HREF=”mailto:NAME”>
<TSSCRIPT value=Contents>
</TSSCRIPT>
</A>
</TSSCRIPT>
<TSSCRIPT value=”EmbedAfterControl” type=html>
</TSSCRIPT>
<!— email.string.htm — End —>
</BODY>
</HTML>
```

Hyperlink String with terse text displayed

This skeleton formats data from a variable containing a URLaddress so it displays as a hyperlink. To use this skeleton, you would specify the hyperlink capability property in the Capabilities prompt and the terse style property in the Style prompt in the Internet Connect template in the IDE (individual overrides for a control).

```
<HTML>
<head>
<meta name=”ts-control” content=”sstring”>
<meta name=”ts-capabilities” content=”hyperlink”>
<meta name=”ts-style” content=”terse”>
</head>
<BODY>
<!— link.string.htm — Start —>
<TSSCRIPT value=”EmbedBeforeControl” type=html>
</TSSCRIPT>
<TSSCRIPT tag=a attr=href replace=NAME value=Contents>
<A HREF=”http://NAME”>Web Site
</A>
```

```
</TSSCRIPT>
<TSSCRIPT value=”EmbedAfterControl” type=html>
</TSSCRIPT>
<!– link.string.htm – End –>.9
</BODY>
</HTML>
```

Hyperlink String with verbose text displayed

This skeleton formats data from a variable containing a URLaddress so it displays as a hyperlink. To use this skeleton, you would specify the hyperlink capability property in the Capabilities prompt and the verbose style property in the Style prompt in the Internet Connect template in the IDE (individual overrides for a control).

```
<HTML>
<head>
<meta name=”ts-control” content=”sstring”>
<meta name=”ts-capabilities” content=”hyperlink”>
<meta name=”ts-style” content=”verbose”>
</head>
<BODY>
<!– link.string2.htm – Start –>
<TSSCRIPT value=”EmbedBeforeControl” type=html>
</TSSCRIPT>
<TSSCRIPT tag=a attr=href replace=NAME value=Contents>
<A HREF=”http://NAME”>
<TSSCRIPT value=Contents>
</TSSCRIPT>
</A>
</TSSCRIPT>
<TSSCRIPT value=”EmbedAfterControl” type=html>
</TSSCRIPT>
<!– link.string2.htm – End –>
</BODY>
</HTML>
```

5 - Skeleton Guide

Introduction

When using the Web Builder templates, special HTML files, called skeletons are used. As the name implies, these files have very little information in them, in other words, they are a “bare bones” template. Theare are used to merge with the Clarion application representation to create an HTML page. Their purpose is to produce HTML code for a single window control, window and application. The only exception is the Window.htm which produces HTML code for the basic page.

These files contain a special scripting language known as TSSCRIPT. For those familiar with scripting languages, it has similar characteristics with JavaScript and XML, although it is not a complete version of either of these. You could also think of it as “templates” for HTML code. The runtime routines read attributes of TSSCRIPT tags. HTML page generation is done on the server when it generates the hard HTML file that is piped to the client. The effect is favorable as it means lower bandwidth usage than Java and a reliable way to predict how a page and its contents are rendered.

Skeletons can include other skeletons as the need arises. The benefit here is that you could author your own skeletons and include them with the shipping skeletons.

Where are the Skeleton files?

As shipped, the skeleton files are located in the *Distrib\Skeleton* folder. Under this folder are three style folders, *Default*, *Fish* and *Wire*. These are theme folders. For purposes of this chapter, the *Default* theme folder is examined. These are the files you will find in this folder:

`Button.htm`
`Combo.htm`
`Group.htm`
`Hotstring.htm`
`List.htm`
`Panel.htm`
`Radio.htm`
`Sheet.one.htm`
`Sstring.htm`
`Tab.one.htm`
`Toolbar.htm`
`Check.htm`
`Detail.htm`
`Entry.htm`
`Image.htm`
`Menu.htm`
`Query.htm`
`Region.htm`
`Sheet.two.htm`
`String.htm`
`Table.htm`
`Window.htm`
`Box.htm`
`Email.string.htm`
`Grid.htm`
`Item.htm`
`Menubar.htm`
`Prompt.htm`
`Sheet.all.htm`
`Spin.htm`
`Tab.all.htm`
`Text.htm`
`Splash.htm`

Window.HTM

This is the main skeleton. This controls the default look or appearance for all windows in your application. This skeleton controls the defaults (which can be overridden later).

Tip

It is recommended that while you are studying the skeletons with this reference, you open them with any text editor, preferably one that understands HTML commands such as TextPad.

Note:

This discussion (and the ones that follow) will work from the top of the skeleton files down.

```
<TSSCRIPT value="EmbedMetaTags" type=html></TSSCRIPT>
```

The <TSSCRIPT> is the beginning tag for using any TSSCRIPT language. It requires the end </TSSCRIPT> tag. Everything in between these two tags are attributes. <TSSCRIPT> begins a section of code that is replaced with HTML at runtime.

The above is an embed point for HTML embedded code in the skeletons.

```
<meta name="ts-control" content="window,application">
```

This is standard HTML declaring HTTP meta name/value pairs that are associated with the HTML document. This is declaring a new meta name called ts-control and it is used in a window or application.

```
<TSSCRIPT include="TimeOut != 0">
  <TSSCRIPT tag=meta attr=content replace="DELAY" value="TimeOut">
    <TSSCRIPT tag=meta attr=content replace="PROGRAM.TARGET"
value="ProgramReference">
<meta HTTP-EQUIV="REFRESH" CONTENT="DELAY;URL=PROGRAM.TARGET">
    </TSSCRIPT>
  </TSSCRIPT>
</TSSCRIPT>
```

This shows several things. First, you can embed HTML code within TSSCRIPT tags.

The *include* attribute on the first line means the code following is used only if the timeout value is not zero. This is very similar to JavaScript. The “!=” means “not equal to”.

The next line replaces the *DELAY* attribute with the value in *TimeOut*. This is set in the global web template for the application. The default is 600 seconds or 10 minutes.

The next line takes the PROGRAM.TARGET attribute and replaces it with the value in ProgramReference. This is the name of the application.

```
<TSSCRIPT value="Text" patch="*" comment="patch title" >
```

This line sets up when the value Text is replaced or patched. The asterisk means replace in all occurrences. This affects the text placed on the caption or title bar.

A few lines down you will see:

```
<TSINCLUDE name="script.htm">
```

This tag inserts the SCRIPT.HTM file containing the JavaScript used within the WebBuilder HTML forms. It does not need an end tag.

The following tags set up the default colors for different controls that can be placed on a page, in other words, it modifies the HTML for controls by setting color attributes, with a default value, to the control's HTML code:

```
<TSSCRIPT comment="Change the colors in the following lines to change the
colors of the generated application"></TSSCRIPT>
<TSSCRIPT tag="<* FinalColor=Border>"       attr=bgcolor
value="'#dcdcdc'" comment="Border Color" phase=*>
<TSSCRIPT tag="<* FinalColor=Header>"       attr=bgcolor
value="'#a0b8c8'" comment="Header Color" phase=*>
<TSSCRIPT tag="<* FinalColor=HeaderB>"      attr=bgcolor
value="'#ccccff'" comment="Header Background Color" phase=*>
<TSSCRIPT tag="<* FinalColor=Cell>"         attr=bgcolor
value="'#ffffcc'" comment="Cell Color" phase=*>
<TSSCRIPT tag="<* FinalColor=CellB>"        attr=bgcolor
value="'#ffffff'" comment="Cell background color" phase=*>
<TSSCRIPT tag="<* FinalColor=DisabledText>" attr=color   value="'Gray'"
comment="Disabled text color" phase=*>
<TSSCRIPT tag="<* FinalColor=HiLightCellColor>" attr=bgcolor
value="'Yellow'"   comment="Highlight cell color" phase=*>
<TSSCRIPT tag="<* FinalColor=HiLightTextColor>" attr=color
value="'Black'"    comment="Highlight text color" phase=*>
<TSSCRIPT tag="<* FinalColor=*>" attr=FinalColor remove phase=Runtime
comment="Remove pseudo tags from the table entries">
```

The lines above should be self-evident (and they are explained in detail in the *Common Questions and Answers* section). The last line is what is interesting.

This is taking whatever attributes for color are used and replacing them with the defaults set in the preceeding tag sets. It does this at *runtime* as the *remove* phase suggests.

These two tag sets determine where images and public pages belong relative to the running program:

```
<TSSCRIPT tag=img attr=src replace="IMAGES" value="Public" allowblank
comment="Correct the path to images (the public directory)">
<TSSCRIPT tag=img attr=src replace="PUBLIC" value="Public" allowblank
comment="Correct the path to the public directory">
```

These two replace attributes are filled in at runtime with the correct reference to the PUBLIC folder. It differs between the linked in broker and a live deployment, since they usually are in different folders.

This means images must be in the PUBLIC folder and are referenced by “/AnImage.GIF” or “/SubFolder/MyImage.GIF”.

The body tag introduces the body of the document. Look at this line:

```
<body finalcolor="Page" bgcolor="white" onload="onBodyLoad()"
onunload="onBodyUnload()">
```

This says that each page loaded gets a white background. The *onload* event occurs before the user agent (the browser) draws anything. The parameter is a script. The *onunload* event whenever an action is taken that will change the current target such as a link, HTML form completion or browser close. Again, the parameter is a script. Both are found in SCRIPTS.HTM.

```
<TSSCRIPT tag=form attr=action value="ProgramReference">
<TSSCRIPT tag=form attr=method value="FormMethod">
<TSSCRIPT tag=form attr=enctype value="FormEncoding">
```

The lines above set up the attributes for the next line of code that begins the HTML form used for all WebBuilder pages. All actions returned to your application are done so through this HTML form or by direct JavaScript SUBMIT().

```
<form name="ClarionForm" method="GET" action="PROGRAM.TARGET"
onsubmit="return (submitSuppress-- == 0);">
```

The next line creates a hiddenHTML form control with a value:

```
<input type="HIDDEN" name="__Special__" value>
```

The source below begins an HTML <table>, </table> tag set.

All HTML generated as a representation of your Clarion procedure window is enclosed within HTML tables. This provides a method to handle placement of controls and text for display within a browser:

```
    <table finalcolor="Border" border="0" cellpadding="4" cellspacing="2"
width="100%">
      <tr finalcolor="Header">
        <td width=99%><b>
          <TSSCRIPT value=Title>
            Page Title
          </TSSCRIPT>
        </b></td>
```

Embedded within the HTML tables, used throughout the skeletons for control placement, you will find other tag sets such as:

<th>, </th>	**Denote a table header row.**
<tr>, </tr>	**Denote a table row.**

Within these table row tags you will find <td>,</td> tag pairs. These tags create the individual cells within a table row. The "d" in "td" is for data.

The HTML source above defines the top row of cells that represent the titlebar of your procedure window. This table row uses the predefined "Header" color, as discussed earlier, for the background for the "Page Title." "Page Title" is the text you display in your procedure titlebar.

```
        <td width=1%>
          <TSSCRIPT tag=a attr=href replace="NAME" value="Name">
          <a href="javascript:icSubmit('NAME$EventCloseWindow');"><img
alt="Close" WIDTH="18" HEIGHT="15" SRC="PUBLIC/x.gif" BORDER=0></a>
          </TSSCRIPT>
        </td>
```

The <td> tag is a table data cell. The width of the cell is expressed as a percent of available space. The percent means to use the smallest space possible, but if more is needed, then the size will grow as needed. The <a> anchor tag is defining an href to some JavaScript for event processing. In other words, if this image is clicked, a close window event is signaled.

```
<tr>
  <td colspan="2">
    <img name="ZONE:Menubar" alt="Wizatrons will place menus in here">
  </td>
</tr>
```

The colspan attribute attribute specifies the number of columns spanned by the current cell. The image name is set to *ZONE:Menubar*, meaning that this column will contain the menu items.

The rest of the skeleton is code covered in the preceeding text, but with different settings and the required end tags.

Script.htm

This skeleton sets up the needed JavaScript functions. It is found in the *Skeleton* sub folder. If you recall in the previous section for the window skeleton, there is a line that says:

```
<TSINCLUDE name="script.htm">
```

This is a TSSCRIPT command to include another file. This file contains JavaScript. Look at the first line:

```
<SCRIPT type="text/JavaScript">
```

The <SCRIPT> tag is HTML. This introduces or starts a script. The *type* attribute is there as there isn't a standard for the *language* attribute. In this script, the *text/JavaScript* is the standard content type for JavaScript. Other examples of content types include *text/html*, *image/png*, *image/gif*, *video/mpeg*, *text/css*, and *audio/basic*.

The next line begins an HTML comment that surrounds all the JavaScript commands and functions:

```
<!-- Hides script from old browsers
```

The end of this HTML comment can be found at the bottom of this file just above the last line. This may seem confusing at first, but the key to understand why this works is that you are working with *two* different languages. The <!-- is the HTML comment. JavaScript comments start with double slashes (//) or slash-asterisk (/*) for multi-line comments. Thus, this entire file is ignored by browsers that cannot handle JavaScript.

Also, you will notice the // JavaScript comment characters are also commenting the end HTML comment. This is because JavaScript will try to interpret the --> characters and it can't. It will result in JavaScript errors if left off.

What is between these comments is the actual JavaScript used in the skeletons. If you wish to add your JavaScript functions, simply add them to this file. We recommend that you add a comment or two if you do.

The purpose of this chapter is not to teach you JavaScript. Since there are JavaScript functions listed throughout the skeletons, it is worth noting where you may find them.

If you view source while your application is running, you will see all the JavaScript in the generated HTML file.

Box.htm

This is a small skeleton. Its sole purpose is to draw a box, or represents a BOX control. But where is the box drawn? And around what? Lets examine some code:

```
<TSSCRIPT tag=table attr=bgcolor value="FillColor" first>
```

Some TSSCRIPT to define table background color attributes.

```
<table border=2>
  <tr>
    <td>
      <img width="300" height="200" name="ZONE:Contents"
alt="Wizard will place controls in here">
    </td>
  </tr>
</table>
```

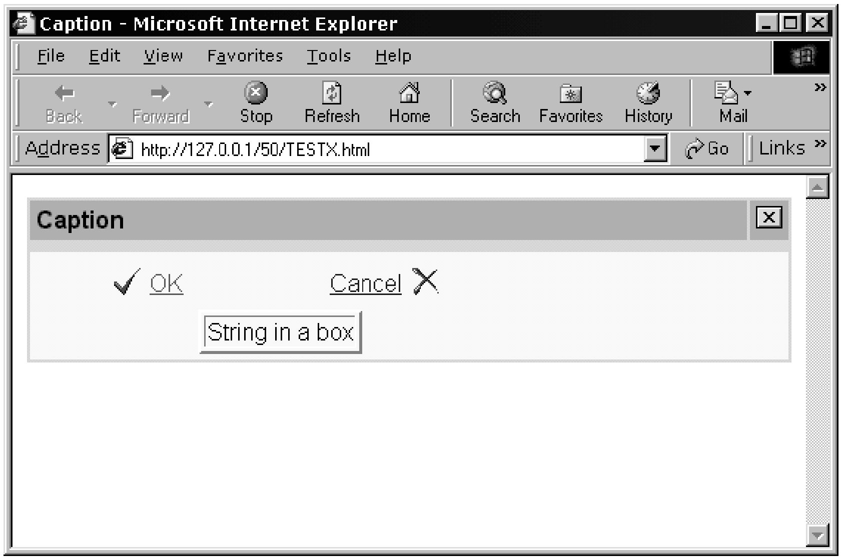
These lines do the magic. The first defines the width of the border. In this case, it is 2 pixels wide, all around the table.

For each table row (<tr>) there is one table data cell <td>). In it is used the image tag with a fixed height and width. It is replaced at HTML generation time with values based on elements that would make up a table. The above is sandwiched in TSSCRIPT tags.

The other lines are discussed in the *window.htm* section. This skeleton is used when you use a BOX control on your window. The generated HTML code appears like this:

```
<!-- Box.htm -- Start -->
<table border=2>
  <tr>
    <td>
  String in a box
    </td>
  </tr>
</table>
<!-- Box.htm -- End -->
```

And this is what it looks like at runtime:


Caption - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address http://127.0.0.1/50/TESTX.html
Go Links
Caption
OK
Cancel
String in a box

Button.htm

This is the skeleton that controls the look and feel of buttons. This skeleton actually has two sections. These sections start with these lines:

```
<TSSCRIPT include="Icon != ''">
<TSSCRIPT omit="Icon != ''">
```

These lines are can be read as:

"Include this section of code if ‘Icon’ is NOT blank." In other words, if this button includes an image, include the text between this TSSCRIPT tag and its ending </TSSCRIPT> tag.

"Omit this section of code if ‘Icon’ is NOT blank." In other words, if this button includes an image, omit the text between this TSSCRIPT tag and its ending </TSSCRIPT> tag.

The next series of code (starting with the section that has images on buttons) has these TSSCRIPT lines:

```
<TSSCRIPT include="Disabled">
  <TSSCRIPT tag=input attr=src value="Image">
    <INPUT type='image' ALT='Disabled' SRC="SRC">
  </TSSCRIPT>
</TSSCRIPT>
```

This is script to have a placeholder for disabled buttons. It simply replaces the attributes of a particular button with its actual attributes.

The next line of script is for buttons that are not disabled. This is done starting with this line:

```
<TSSCRIPT omit="Disabled">
```

The script in this section simply checks for image placement, either left or right justified and writes the appropriate HTML code, including the spacing of the button, which is placed in a <table>. If there is no text on a button (image only), then the button is rendered accordingly.

The remaining script deals with text only buttons.

This skeleton is used for all button controls on a page and calls the JavaScript functions to process the button. In the case of text only buttons, it uses the HTML submit attribute for input. No JavaScript is required in this case.

Here is the resulting HTML code generated with a window with two buttons, one left justified, the other right:

```
<TABLE cellpadding=0 cellspacing=0 border=0 WIDTH=100%><TR><TD
WIDTH="8%"></TD>
<TD WIDTH="29%" COLSPAN=2>
       <table cellspacing=0 cellpadding=0><tr>
       <td>
       <a href="javascript:icSubmit('OKBUTTON');"><img
SRC='/51/wizok.gif' BORDER=0 alt=OK></a>
       </td>
       <td>
       <a href="javascript:icSubmit('OKBUTTON');">OK</a></td>
       </tr></table>
</TD>
<TD WIDTH="7%"></TD>
<TD WIDTH="29%">
       <table cellspacing=0 cellpadding=0><tr>
       <td><a href="javascript:icSubmit('CANCELBUTTON');">Cancel</a></td>
       <td>
       <a href="javascript:icSubmit('CANCELBUTTON');"><img
SRC='/51/wizcncl.gif' BORDER=0 alt=Cancel></a>
       </td>
       </tr></table>
</TD>
```

Check.htm

This is the check box skeleton. When you use a CHECK control, this skeleton is used to generate the HTML code for it.

```
<TSSCRIPT include="Disabled">
  <font FinalColor=DisabledText>
    <TSSCRIPT include="Checked">
      [X]
    </TSSCRIPT>
    <TSSCRIPT omit="Checked">
      [ ]
    </TSSCRIPT>
    <TSSCRIPT value=DisplayText>
    Checkbox text
    </TSSCRIPT>
  </font>
</TSSCRIPT>
```

The above handles the disabled check boxes, whether they are checked or not. Notice that it includes the default colors from the window skeleton.

The enabled checkbox uses JavaScript to sumbit the actions for the control:

```
<TSSCRIPT omit="Disabled">
  <TSSCRIPT tag=input attr=name value="Name">
  <TSSCRIPT tag=input attr=id value="Name">
  <TSSCRIPT tag=input attr=checked when="Checked">
  <TSSCRIPT tag=input attr=onClick text="icSubmitForm()"
when="SubmitOnChange">
    <input type="checkbox" value="1">
    <TSSCRIPT tag=label attr=for value="Name">
      <label for="above">
      <TSSCRIPT value=DisplayText>Checkbox text</TSSCRIPT>
      </label>
    </TSSCRIPT>
  </TSSCRIPT>
  </TSSCRIPT>
  </TSSCRIPT>
  </TSSCRIPT>
</TSSCRIPT>
```

The reason for the difference is that disabled controls do not generate events, thus it is overkill to have JavaScript render it when HTML is fine.

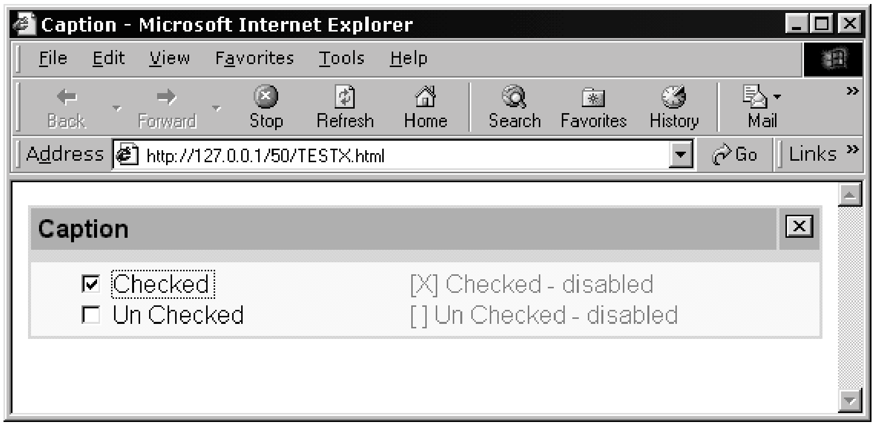
This is the HTML code generated at runtime:

```
<TD WIDTH="43%">
  <!-- Check.htm -- Start -->
    <input type="checkbox" value="1" name=CHECK1 id=CHECK1>
      <label for='CHECK1'>Checked
      </label>
  <!-- Check.htm -- End -->
</TD>
<TD WIDTH="48%">
  <!-- Check.htm -- Start -->
  <font color=Gray>
      [X]

    Checked - disabled
  </font>
  <!-- Check.htm -- End -->
</TD>
<TD WIDTH="4%"></TD>
</TR><TR><TD WIDTH="5%"></TD>
<TD WIDTH="43%">
  <!-- Check.htm -- Start -->
    <input type="checkbox" value="1" name='CHECK1_2'
id='CHECK1_2'>
      <label for='CHECK1_2'>Un Checked
      </label>
  <!-- Check.htm -- End -->
</TD>
<TD WIDTH="48%">
  <!-- Check.htm -- Start -->
  <font color=Gray>
      [ ]

    Un Checked - disabled
  </font>
  <!-- Check.htm -- End -->
</TD>
```

And this is how it looks in a browser:


Caption - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address http://127.0.0.1/50/TESTX.html Go Links

Caption

☑ Checked [X] Checked - disabled
☐ Un Checked [ ] Un Checked - disabled

Combo.htm

This is the skeleton for COMBO controls.

Detail.htm

This is used when making shopping cart applications.

Email.String.htm

This skeleton is used to make an anchor tag (<a>) with an *href* attribute of *mailto:<EmailAddress>*. The EmailAddress needs a properly formatted email address and parameters.

Entry.htm

This skeleton is used for ENTRY controls. For each entry control populated on a window, the skeleton produces the correct HTML code for the entry. It incorporates the attributes for the control.

If an entry control is read-only, then these skeleton code takes care of that:

```
<TSSCRIPT include="Disabled || ReadOnly">
  <TSINCLUDE Name="displayText.htm">
</TSSCRIPT>
```

This just uses a different skeleton for these types of controls, they just display them as text.

This section is for entry controls that are not read-only or disabled:

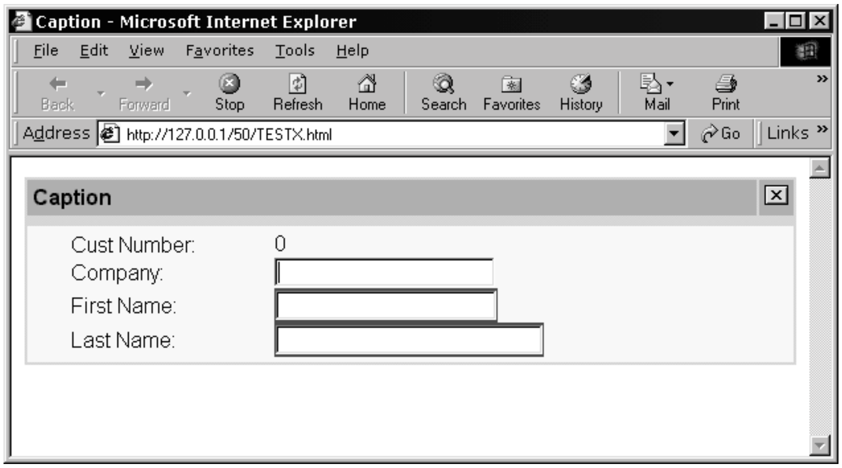
```
<TSSCRIPT omit="Disabled || ReadOnly">
  <TSSCRIPT tag=input attr=name value="Name">
  <TSSCRIPT tag=input attr=value value="DisplayText">
  <TSSCRIPT tag=input attr=type text="Password" when="Password">
  <TSSCRIPT tag=input attr=size value="(Width+2)/4">
  <TSSCRIPT tag=input attr=onChange text="icSubmitForm()"
when="SubmitOnChange">
  <TSSCRIPT tag=input attr=onFocus text="this.select()"
when=SelectOnFocus>
```

The above simply gets the entry name, its prompt text, password type entries (if applicable), default width and set up the JavaScript to detect the event when it is selected.

```
<TSSCRIPT include="Req">
      <table border="0" bgcolor="#FF0000" cellspacing="1" cellpadding="0">
        <tr><td>
          <input type=text>
        </td></tr>
      </table>
    </TSSCRIPT>
    <TSSCRIPT omit="Req">
      <input type=text>
    </TSSCRIPT>
  </TSSCRIPT>
  </TSSCRIPT>
  </TSSCRIPT>
  </TSSCRIPT>
  </TSSCRIPT>
  </TSSCRIPT>
</TSSCRIPT>
```

The next section places a red border around the entry control if the entry is required. The other attrbutes describe how thick the border is, how far around the entry control the border is and any padding.

Here is what a entry page could look like:


Caption - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address
Go
Links

http://127.0.0.1/50/TESTX.html

Caption

Cust Number: 0
Company:
First Name:
Last Name:

And this is the HTML code for the company name in the above example:

```
<!--Entry.htm -- Start -->
  <input type=text name='CUS_COMPANY' size=22 onFocus='this.select()'>
<!--Entry.htm -- End -->
```

The following HTML code is for the required entries:

```
<!--Entry.htm -- Start -->
   <table border="0" bgcolor="#FF0000" cellspacing="1" cellpadding="0">
   <tr><td>
   <input type=text name='CUS_FIRSTNAME' size=22 onFocus='this.select()'>
   </td></tr>
   </table>
<!--Entry.htm -- End -->
```

Grid.htm

This is the grid skeleton, used when a browse grid control is placed on a LIST control.

Group.htm

This is the skeleton used when a GROUP control is populated on a window, Group controls can be used to group related subjects together or they are used with radio buttons.

```
<TSSCRIPT include=Boxed>
  <TSSCRIPT tag=table attr=bgcolor value="BorderColor" first>
  <table width="100%">
    <tr>
      <td>
```

The above first checks to see if the group is boxed. If it is, then it needs to determine the border color of the box. The table width attribute is expressed as a percent of available space, in this case, use all that is available.

The next section simply defines how wide the border is and what the text of the group structure is based on the window control.

```
        <table border="0" width="100%">
```

This next section shows an interesting aspect of TSSCRIPT. In this case, everything that is between the beginning and end tags is replace by real values, but only if there is some text to substitute.

```
          <TSSCRIPT include="DisplayText!=''">
            <tr finalcolor="Header">
              <td><b><TSSCRIPT value=DisplayText>
                Header Text
                </TSSCRIPT>
              </b></td>
            </tr>
          </TSSCRIPT>
```

If the boxed attribute is not set, then this part of the skeletons is used:

```
<TSSCRIPT omit=Boxed>
  <img width="500" height="261" name="ZONE:Contents" alt="Wizatrons will
place controls in here">
</TSSCRIPT>
```

The following HTML code is what the skeletons generate for a simple group:

```
<!--Group.htm -- Start -->
  <table width="100%">
    <tr>
      <td>
        <table border="0" width="100%">
            <tr bgcolor='#a0b8c8'>
              <td><b>
                Group 1
              </b></td>
            </tr>
            <tr>
              <td>
  String in group one</td>
            </tr>
        </table>
      </td>
    </tr>
  </table>
<!--Group.htm -- End -->
```

And this is for the radio groups:

```
<!--Group.htm -- Start -->
  <table width="100%">
    <tr>
      <td>
        <table border="0" width="100%">
            <tr bgcolor='#a0b8c8'>
              <td><b>
                Group Two
              </b></td>
            </tr>
            <tr>
              <td>
<TABLE cellpadding=0 cellspacing=0 border=0><TR><TD WIDTH="13%"></TD>
<TD WIDTH="48%">
<!-- Radio.htm -- Start -->
  <input type="Radio" name='OPTION1$Choice' id='OPTION1_RADIO1'
value=1><label for='OPTION1_RADIO1'>Radio 1</label>
  <!-- Radio.htm -- End -->
</TD>
<TD WIDTH="39%"></TD>
</TR><TR><TD WIDTH="13%"></TD>
<TD WIDTH="48%">
<!-- Radio.htm -- Start -->
  <input type="Radio" name='OPTION1$Choice' id='OPTION1_RADIO2'
value=2><label for='OPTION1_RADIO2'>Radio 2</label>
  <!-- Radio.htm -- End -->
</TD>
```

```
<TD WIDTH="39%"></TD>
</TR></TABLE></td>
            </tr>
        </table>
      </td>
    </tr>
  </table>
<!--Group.htm -- End -->
```

If you notice, it includes the radio skeleton (see the section on radio skeletons). This is what the HTML code looks like at runtime:

Hotstring.htm

This skeleton is designed to be used with the template interface. Notice these two lines:

```
<meta name="ts-control" content="sstring">
<meta name="ts-capabilities" content="hotlink">
```

They align with the two template entry controls, control and capabilities. This creates a link from a string.

Image.htm

This small skeleton handles images on your page.

Take a look at these lines (which is really all there is to this skeleton):

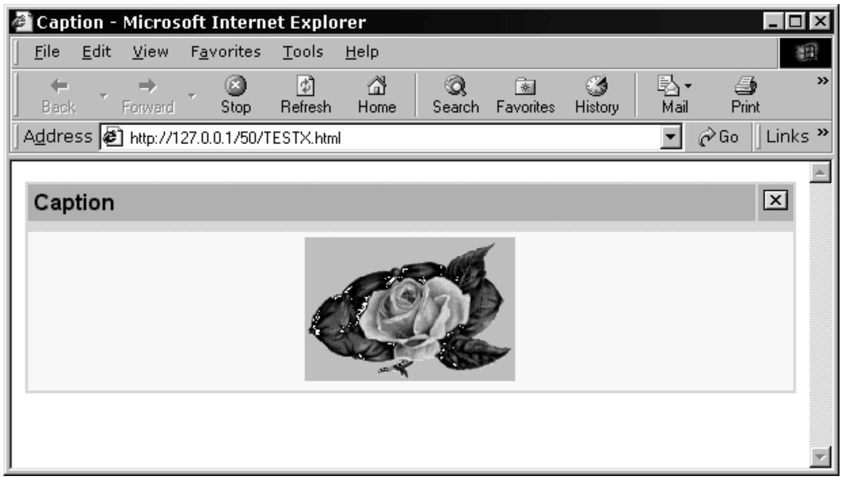
```
<!-- Image.htm -- Start -->
<TSSCRIPT tag=a attr=href value="'javascript:icSubmit(\''+Name+'=1\')'"
when=SubmitOnChange>
<TSSCRIPT tag=img attr=alt value="AltText">
<TSSCRIPT tag=img attr=src value="Image">
<TSSCRIPT tag=img attr=width value="PixelWidth">
<TSSCRIPT tag=img attr=height value="PixelHeight">
<TSSCRIPT tag=img attr=border text="0" when=SubmitOnChange>
<a><img></a>
```

The first line sets up the JavaScript to process an event, such as when a record is changed, thus a refresh of the image is needed.

The next lines set up the ALT text, the actual image, the image height and width (in pixels) and a border. All these are replaced in the last line, like this:

```
<!-- Image.htm -- Start -->
<a><img src='/50/Rose.gif' width=155 height=106></a>
<!-- Image.htm -- End -->
```

And this is what an image looks like in the browser:



Item.htm

This skeleton produces the menu items. If the menu item is a separator, then it produces the HTML tag for horizontal line, <HR>. If the menu item is disabled, it displays the text in the disabled color (see the *Window* skeleton).

If the menu item is clicked, this is detected by the JavaScript icSubmit function (see the *Scripts* skeleton).

List.htm

This sets up a drop list control. The list itself is populated by another skeleton (see the *Select* skeleton).

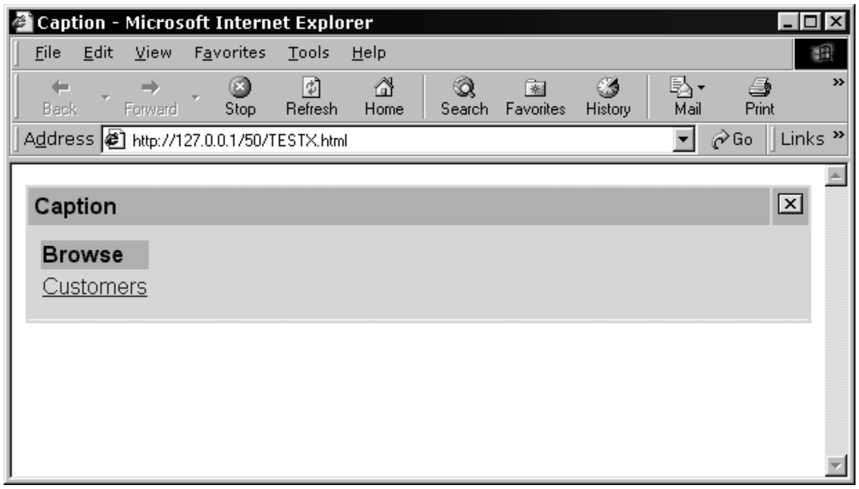
Menu.htm

This skeleton handles the menus.

The generated HTML looks like this:

```
<!--Menubar.htm -- Start -->
<table border=0>
  <tr valign=top>
<td>
 <table border=0 bgcolor='#dcdcdc'>
    <tr bgcolor='#a0b8c8'>
      <td><b>
      Browse
      </b></td>
    </tr>
    <tr><td>
      <a href="javascript:icSubmit('BROWSEBROWSECUSTOMERS');">
        <NOBR>Customers</NOBR>
      </a><br>
    </td></tr>
  </table>
</td>
  </tr>
</table>
<!--Menubar.htm -- End -->
```

The above code looks like this:


Caption - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Forward
Stop
Refresh
Home
Search
Favorites
History
Mail
Print
Address
http://127.0.0.1/50/TESTX.html
Go
Links
Caption
Browse
Customers

This is the skeleton code that produced the above:

```
<!--Menu.htm -- Start -->
<form>
<TSSCRIPT value="EmbedBeforeControl" type=html>
</TSSCRIPT>
<td>
  <table FinalColor=Border border=0>
    <tr FinalColor=Header>
      <td><b>
      <TSSCRIPT value=DisplayText>
      <p>
      This is the text
      </p>
      </TSSCRIPT>
      </b></td>
    </tr>
    <tr FinalColor=CellColor><td>
   <img name="ZONE:Contents" alt="Wizatrons will place controls in here">
   </td></tr>
  </table>
```

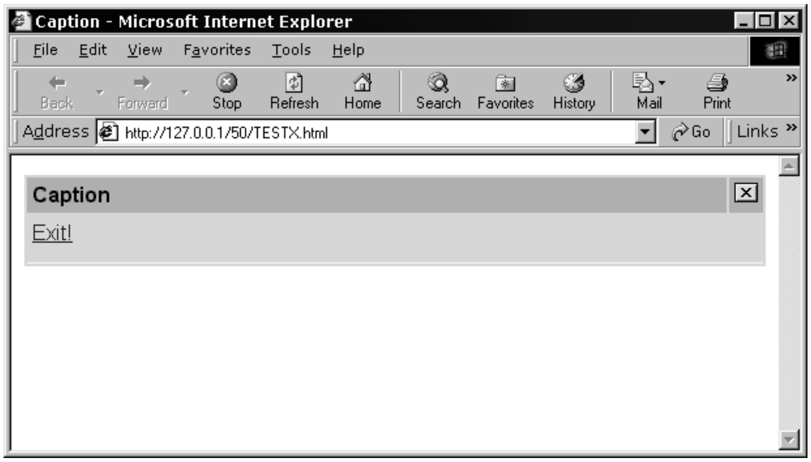
```
</td>
<TSSCRIPT value="EmbedAfterControl" type=html>
</TSSCRIPT>
</form>
<!--Menu.htm -- End -->
```

Menubar.htm

This skeleton is little different than menu. It is used only when you do not have a drop menu, but a menu bar with items only. The following HTML code is generated at runtime:

```
<!--Menubar.htm -- Start -->
<table border=0>
  <tr valign=top>
      <a href="javascript:icSubmit('EXIT');">
        <NOBR>Exit!</NOBR>
      </a><br>
  </tr>
</table>
<!--Menubar.htm -- End -->
```

This is what is looks like in the browser:


Caption - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address http://127.0.0.1/50/TESTX.html
Go Links
Caption
Exit!

The skeleton that produced it is as follows:

```
<!--Menubar.htm -- Start -->
<TSSCRIPT value="EmbedBeforeControl" type=html>
</TSSCRIPT>
<table border=0>
  <tr valign=top>
    <img name="ZONE:Contents[width=1%]" alt="Wizatrons will place
controls in here">
  </tr>
</table>
```

```
<TSSCRIPT value="EmbedAfterControl" type=html>
</TSSCRIPT>
<!--Menubar.htm -- End -->
```

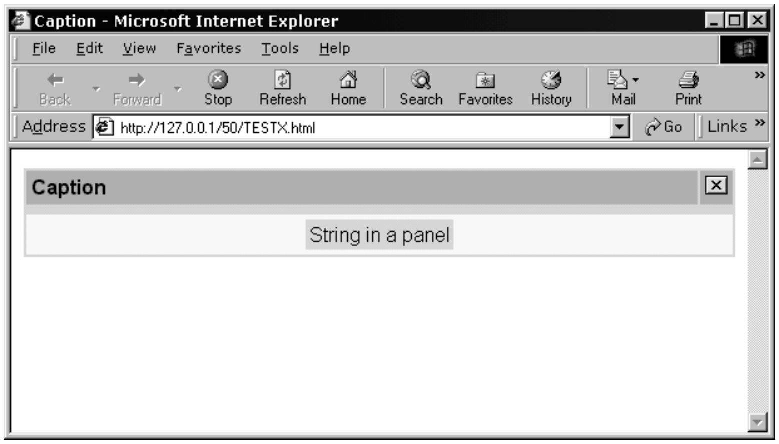
Panel.htm

This skeleton creates panel controls in HTML. The effect is that you can use this control to create nice backgrounds around controls.

The skeleton code is as follows:

```
<table FinalColor=Border>
  <tr>
    <td>
   <table FinalCoor=Header border="0" cellpadding="0" cellspacing="0" width="100%">
        <tr>
        <td><img width="300" height="200" name="ZONE:Contents"
alt="Wizatrons will place controls in here"></td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```

Which makes it look similar to this:



http://127.0.0.1/50/TESTX.html

Caption

String in a panel

The actual HTML code is as follows:

```
<!-- Panel.htm -- Start -->
<head>
</head>
<table bgcolor='#dcdcdc'>
  <tr>
    <td>
```

```
      <table FinalCoor=Header border="0" cellpadding="0" cellspacing="0"
width="100%">
        <tr>
        <td>
  String in a panel</td>
        </tr>
      </table>
    </td>
  </tr>
</table>
<!-- Panel.htm -- End -->
```

If you notice, there is no TSSCRIPT for the panel, yet it uses “variables” set by TSSCRIPT commands. These come from the window skeleton.

Prompt.htm

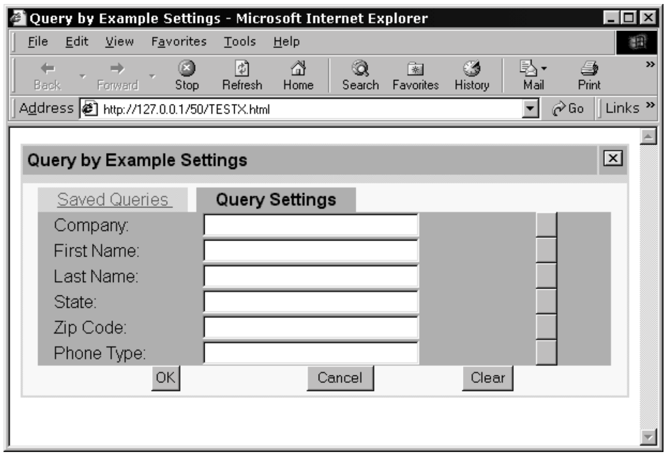
The Prompt skeleton includes the *DisplayText* skeleton to do its work. Other than two embed points, that is all there is. See *DisplayText*.

Query.htm

The Query skeleton is used for QBE controls and is used when a user is performing QBE functions. The HTML produced is as follows:

```
<!-- Start of Query.htm -->
    <input type="HIDDEN" value='0' name='Feq1020$Choice'>
  <input type="SUBMIT" value='    ' name='Feq1020'
onClick='cycleQuery(this, ClarionForm.Feq1020$Choice)'>
<!-- End of Query.htm -->
```

This code is produced for each control in the query dialog. The QBE template default is for a form interface and this is how it is rendered in a browser:



Note:

You cannot use the list version of QBE in a web application as it uses edit-in-place. You will get a warning message about this if you do.

The buttons to the right of the entries set the matching rules (greater than, less than, etc) and work just like the desktop version. Each press of these buttons cycles through all the valid choices. This is done by these lines of skeleton code:

```
<TSSCRIPT tag=input attr=onClick replace="NAME" value="Name">
  <input type="SUBMIT" value="    " name="NAME" onClick="cycleQuery(this,
ClarionForm.NAME$Choice)">
</TSSCRIPT>
```

It calls a JavaScript function called *cycleQuery*. This function is defined in the *scripts.htm* file. The function is simple and if you examine the code, it is not too dissimilar to the way it would be coded in Clarion:

```
  function cycleQuery(cur, choice)
  {
    submitSuppress++;
    choice.value = (Number(choice.value) + 1) % 5
    switch (Number(choice.value))
    {
    case 0: cur.value = '    '; break;
    case 1: cur.value = ' = '; break;
    case 2: cur.value = '>='; break;
    case 3: cur.value = '<='; break;
    case 4: cur.value = '<>'; break;
    }
  }
```

The *switch* command is the same as the CASE in Clarion. The *case* is the same as the OF.

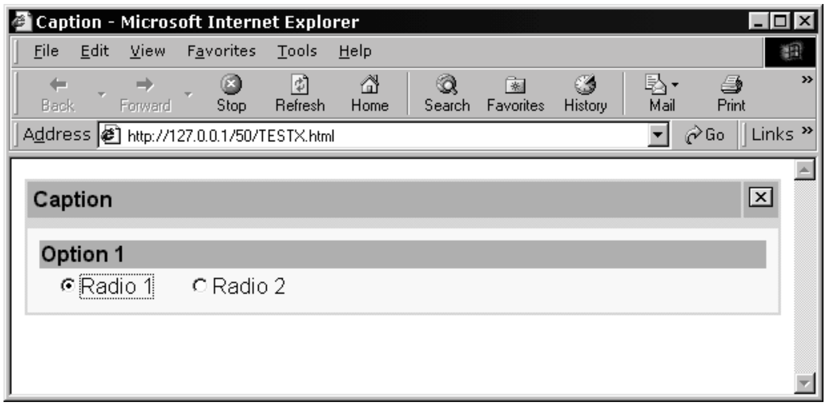
Radio.htm

This is the radio button skeleton. It is used when radio buttons are displayed. While radio buttons are used on lists to indicate the highlighted row, this is done with another skeleton. See Table.htm.

For option groups (groups that contain radio buttons), the following is the HTML generated for each radio button:

```
<!-- Radio.htm -- Start -->
  <input type="Radio" name='OPTION1$Choice' id='OPTION1_RADIO1'
value=1><label for='OPTION1_RADIO1'>Radio 1</label>
 <!-- Radio.htm -- End -->
```

It looks like this when running in a browser:



The skeleton code for generating HTML radio buttons is as follows:

```
<TSSCRIPT omit="Disabled">
  <TSSCRIPT tag=input attr=id value="Name">
  <TSSCRIPT tag=input attr=Name replace="NAME" value="Container.Name">
  <TSSCRIPT tag=input attr=checked when="Container.ChoiceFEQ==FEQ">
  <TSSCRIPT tag=input attr=disabled when="Disabled">
  <TSSCRIPT tag=input attr=onClick text="icSubmitForm()"
when="SubmitOnChange || Container.SubmitOnChange">
  <TSSCRIPT tag=input attr=value value="ChildIndex">
  <input type="Radio" name="NAME$Choice">
  <TSSCRIPT tag=label attr=for value="Name">
    <label>
      <TSSCRIPT value=DisplayText>
      </TSSCRIPT>
    </label>
  </TSSCRIPT>
```

The above is used only when the radio button is enabled. If disabled, it includes the DisplayText skeleton.

The various TSSCRIPT lines gather information about the radio button, including it’s Field Equate and setting up event handling via a JavaScript function.

This is so that the HTML code that declares an input of radio type, also gives a name to this control. As you can see in the generated HTML, the TSSCRIPT commands above build the input tag.

Region.htm

This skeleton is used for REGION controls.

Sheet.all.htm

Not documented at this time.

Sheet.one.htm

This is the default skeleton used when a sheet control is used. Even with sheet controls containing other sheet controls on your window, this skeleton will generate the HTML code to render it in your browser.

This skeleton does not do much as the code below shows:

```
<!-- Sheet One.htm -- Start -->
<TSSCRIPT value="EmbedBeforeControl" type=html></TSSCRIPT>
<IMG name="ZONE:Default:Contents" alt="Wizatrons will place controls in
here">
<TSSCRIPT value="EmbedAfterControl" type=html></TSSCRIPT>
<!-- Sheet One.htm -- End -->
```

The window skeleton causes the tab skeleton to be included in the sheet. See *Window* and *Tab* skeleton sections.

Sheet.two.htm

Not documented at this time.

Spin.htm

This skeleton gathers data and attributes about spin controls on a window and renders the HTML code to produce the control, complete with event trapping.

There are two major sections (if the control is enabled, otherwise it uses the DisplayText skeleton to represent a disabled control - See *DisplayText* skeleton).

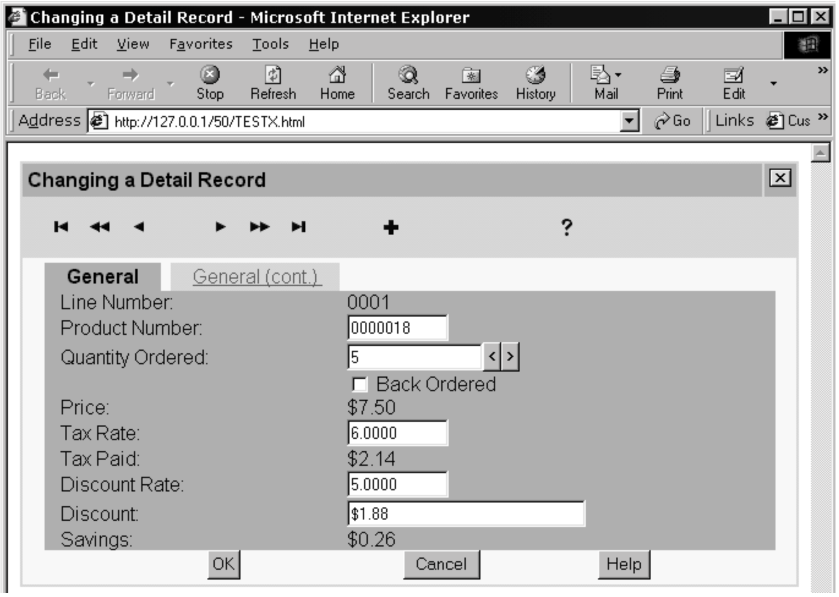
If the spin control does not have a From entry, this section of skeleton code is used:

```
  <TSSCRIPT Omit="From!=''">
    <table cellpadding="0" cellspacing="0"><tr><td>
    <TSSCRIPT tag=input attr=value value="DisplayText">
    <TSSCRIPT tag=input attr=name value="Name">
    <TSSCRIPT tag=input attr=size value="(Width+2)/4">
    <input type="TEXT" value="Text" name="NAME">
    </TSSCRIPT>
    </TSSCRIPT>
    </TSSCRIPT></td>
    <TSSCRIPT tag=input attr=onClick replace="NAME" value="Name">
    <TSSCRIPT tag=input attr=onClick replace="RANGEHIGH"
value="RangeHigh">
    <TSSCRIPT tag=input attr=onClick replace="RANGELOW" value="RangeLow">
    <TSSCRIPT tag=input attr=onClick replace="STEP" value="Step">
    <td><input type="submit" value="&lt;"
onclick="spin(ClarionForm.NAME,-STEP,RANGEHIGH,RANGELOW);"></td>
    <td><input type="submit" value="&gt;"
onclick="spin(ClarionForm.NAME,+STEP,RANGEHIGH,RANGELOW);"></td>
    </TSSCRIPT>
    </TSSCRIPT>
    </TSSCRIPT>
    </TSSCRIPT>
    </tr></table>
  </TSSCRIPT>
```

This is the generated HTML code:

```
<!-- Spin.htm -- Start -->
 <table cellpadding="0" cellspacing="0"><tr><td>
 <input type="TEXT" value='5.00' name='DTL_QUANTITYORDERED' size=14></td>
 <td><input type="submit" value="&lt;"
onclick='spin(ClarionForm.DTL_QUANTITYORDERED,-1,999,1);'></td>
 <td><input type="submit" value="&gt;"
onclick='spin(ClarionForm.DTL_QUANTITYORDERED,+1,999,1);'></td>
</tr></table>
<!-- Spin.htm -- End -->
```

The above will produce a spin box like the following image:


Changing a Detail Record - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://127.0.0.1/50/TESTX.html
Go Links Cus
Changing a Detail Record
?
General
General (cont.)
OK
Cancel
Help

Line Number:	0001
Product Number:	0000018
Quantity Ordered:	5 < >
	☐ Back Ordered
Price:	$7.50
Tax Rate:	6.0000
Tax Paid:	$2.14
Discount Rate:	5.0000
Discount:	$1.88
Savings:	$0.26

If the From entry is used (meaning it gets its values from a Queue), the following skeleton code is used to produce the HTML code at runtime:

```
<TSSCRIPT Include="From!=''">
    <TSSCRIPT tag=input attr=name value="Name">
    <TSSCRIPT tag=input attr=value value="DisplayText">
    <TSSCRIPT tag=input attr=size value="(Width+2)/4">
    <TSSCRIPT tag=input attr=onChange text="icSubmitForm()"
when="SubmitOnChange">
    <TSSCRIPT tag=input attr=onFocus text="this.select()"
when=SelectOnFocus>
      <TSSCRIPT include="Req">
        <table border="0" bgcolor="#FF0000" cellspacing="1"
cellpadding="0">
          <tr><td>
            <input type=text>
          </td></tr>
        </table>
      </TSSCRIPT>
      <TSSCRIPT omit="Req">
        <input type=text>
      </TSSCRIPT>
    </TSSCRIPT>
    </TSSCRIPT>
    </TSSCRIPT>
    </TSSCRIPT>
    </TSSCRIPT>
  </TSSCRIPT>
```

Splash.htm

This skeleton produces the HTML for splash procedures. This skeleton is a different version of the window skeleton, as splash procedures usually have a different look and feel than the appearance of the rest of the program. To this end, there are settings that are different than the window skeleton.

Tip

If you like the style of the splash skeleton, you can use it as the skeleton for the window of any procedure. You can do this by setting the window override in the local extension.

Only the differences between this skeleton and the window skeleton are covered in this section. See Window.htm for more information.

Outside of some TSSCRIPT differences, the following skeleton code is what makes the difference:

```
<CENTER>
<table bgcolor="#ccccff" border="1" width=60%>
  <tr>
    <td valign="center" align="center">
   <img name="ZONE:Contents" alt="Wizatrons will place controls in here">
    </td>
  </tr>
  <tr>
    <td valign="center" align="center">
      <TSSCRIPT tag=a attr=href value="ProgramReference">
        <A HREF="PROGRAM.TARGET">Continue</A>
      </TSSCRIPT>
    </td>
  </tr>
</table>
</CENTER>
```

Most of the above is cosmetic (colors, alignments, borders, etc.). Inspect the <A HREF> line. This adds a hyperlink that looks like this at runtime:

```
<A HREF='/MYPROGRAM.EXE.80'>Continue</A>
```

Sstring.htm

Not documented at this time.

String.htm

A very simple skeleton that includes another skeleton. See *DisplayText*.

Tab.all.htm

Not documented at this time.

Tab.one.htm

This skeletons shows the tabs on a sheet. Each tab is actually a link with a background color. This skeleton does quite a lot to enure that the tabs work like the program’s desktop equivalent.

```
<!-- Tab One.htm -->
<TSSCRIPT local name=SelectedTab value="phase=='runtime' ?
Container.Choice : ChildIndex">
```

This line sets up the current tab with the current key used for sorting. At runtime, the attributes are replaced by the data in the program.

A few lines down and you see this TSSCRIPT line:

```
<TSSCRIPT omit="Wizard">
```

This means that this procedure is not a Wizard style procedure. If it is, then all remaining skeleton code is not used.

This line sets up a “loop” to process each tab on the procedure:

```
<TSSCRIPT repeat times="Container.NumTabs" name=curTab>
```

This means that for every tab placed on the list, the remaining skeleton code sets up the HTML code to be generated at runtime.

Now examine this code a few lines down:

```
<td FinalColor=Header nowrap>
  &nbsp;&nbsp;&nbsp;
  <b><TSSCRIPT value="thisTab.DisplayText">SELECTEDTAB</TSSCRIPT></b>
  &nbsp;&nbsp;&nbsp;
</td>
```

This code uses some HTML code to space the text on the tabs and ensure they stay on one line This is done with &nbsp, which means “non-breaking space”. If this is not used, the text on tabs could wrap unpredictably. Plus, it ensures that there is white space before and after the text. This makes the text look even on all sides of the tab.

The next section sets up the events for selecting a different tab:

```
    <td FinalColor=Border nowrap>
              <TSSCRIPT tag=a attr=* replace="PROGRAM"
value="ProgramReference">
                <TSSCRIPT tag=a attr=* replace="NAME"
value="Container.Name">
                  <TSSCRIPT tag=a attr=* replace="CURTAB" value="curTab">
      &nbsp;&nbsp;&nbsp;
      <a href="javascript:icSubmit(&quot;NAME$Choice=CURTAB&quot;);">
      <TSSCRIPT value="thisTab.DisplayText">UNSELECTEDTAB</TSSCRIPT>
      </a>
      &nbsp;&nbsp;&nbsp;
                  </TSSCRIPT>
                </TSSCRIPT>
              </TSSCRIPT>
    </td>
```

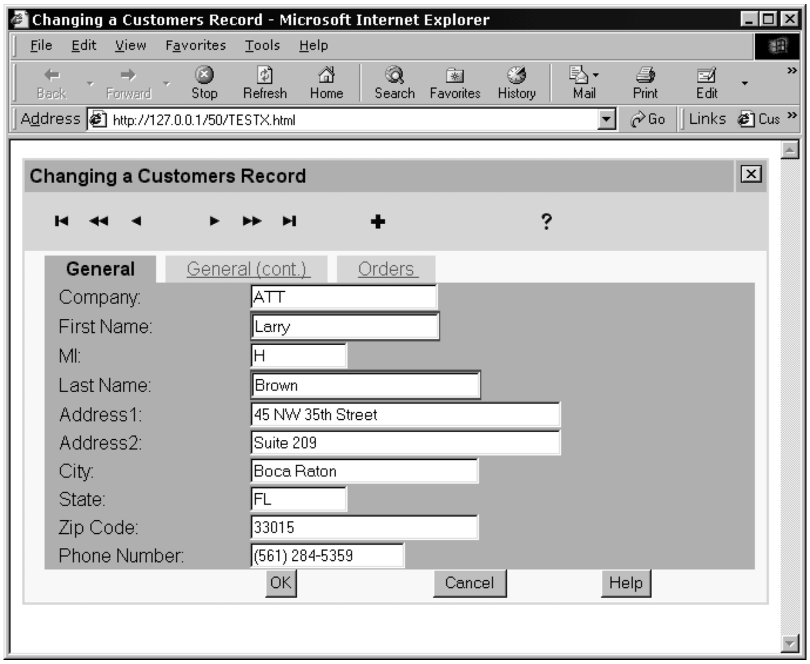
This section takes the values in the program (the TSSCRIPT lines), adds some non-breaking spaces (for tab separation). It then uses a JavaScript function to process the event for when a new tab is chosen.

At runtime, the following HTML is generated (edited for content):

```
<!-- Tab One.htm -->
<table border="0" cellpadding="0" cellspacing="0" width="100%">
  <tr align="left"><td>
    <table border="0" cellpadding="2" cellspacing="0" width="1%"><tr>
    <td nowrap bgcolor='#a0b8c8'>
      &nbsp;&nbsp;&nbsp;
      <b>General</b>
      &nbsp;&nbsp;&nbsp;
    </td>
    <td>&nbsp;</td>
    <td nowrap bgcolor='#dcdcdc'>
      &nbsp;&nbsp;&nbsp;
<a href='javascript:icSubmit(&quot;CURRENTTAB$Choice=2&quot;);'>General
(cont.)
</a>
&nbsp;&nbsp;&nbsp;
    </td>
    <td>&nbsp;</td>
    <td nowrap bgcolor='#dcdcdc'>
      &nbsp;&nbsp;&nbsp;
<a href='javascript:icSubmit(&quot;CURRENTTAB$Choice=3&quot;);'>Orders
</a>
&nbsp;&nbsp;&nbsp;
    </td>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
  </tr>
  </table></td></tr>
  <tr>
    <td bgcolor='#a0b8c8'>
<TABLE cellpadding=0 cellspacing=0 border=0 WIDTH='100%'><TR><TD
WIDTH="2%"></TD>
<TD WIDTH="27%">
  Company:
</TD>
<TD WIDTH="40%" COLSPAN=3>
<!--Entry.htm -- Start -->
!Entry controls here
<!--Entry.htm -- End -->
</TD>
<TD WIDTH="10%"></TD>

<!-- /Tab One.htm -->
```

The following is what it looks like:


Changing a Customers Record - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://127.0.0.1/50/TESTX.html
Go Links Cus
Changing a Customers Record
?
General
General (cont.)
Orders
OK
Cancel
Help

Company: ATT
First Name: Larry
MI: H
Last Name: Brown
Address1: 45 NW 35th Street
Address2: Suite 209
City: Boca Raton
State: FL
Zip Code: 33015
Phone Number: (561) 284-5359

Table.htm

This skeleton handles the way LIST controls on browse procedures are rendered with HTML.

This group of skeleton code sets the text for each column header:

```
  <TSSCRIPT repeat times="FromColumns" name=column>
    <th>
      <TSSCRIPT value="ColumnHeader[column-1]">
HEADERTEXT
      </TSSCRIPT>
    </th>
  </TSSCRIPT>
```

This set of code a little further down, produces the radio buttons that indicate the current row on the list:

```
<TSSCRIPT tag=input attr=name replace="NAME" value="Name">
<TSSCRIPT tag=input attr=value value="row">
<TSSCRIPT tag=input attr=checked value="1" when="Choice==row">
<TSSCRIPT tag=input attr=id replace="FEQ" value="Name">
<TSSCRIPT tag=input attr=id replace="ROWNO" value="row">
```

```
<TSSCRIPT tag=input attr=onClick text="icSubmitForm()"
when="SubmitOnChange">
                          <input type="radio" value="ROW"
name="NAME$Choice" id="FEQ$ROWNO">
```

Further down is the code to format the text that appears in each cell of the list:

```
<TSSCRIPT local name="curColor" value="CellForeColor[row-1][column-1]">
<TSSCRIPT value="'<FONT color=' + MakeColor(curColor) + '>'" type=html
when="curColor !=0"></TSSCRIPT>
<TSSCRIPT tag=label attr=for replace="FEQ" value="Name">
<TSSCRIPT tag=label attr=for replace="ROWNO" value="row">
  <LABEL FOR="FEQ$ROWNO">
<TSSCRIPT value="CellText[row-1][column-1]=='' ? '&nbsp;' : CellText[row-
1][column-1]" phase=runtime>
  CELLTEXT
```

After the closing tags, the navigation buttons are placed at the bottom of the list:

```
<TSSCRIPT include="NavigationControls">
```

The navigation buttons are worthless if there is no event processing for each button. This is done with calls to JavaScript functions:

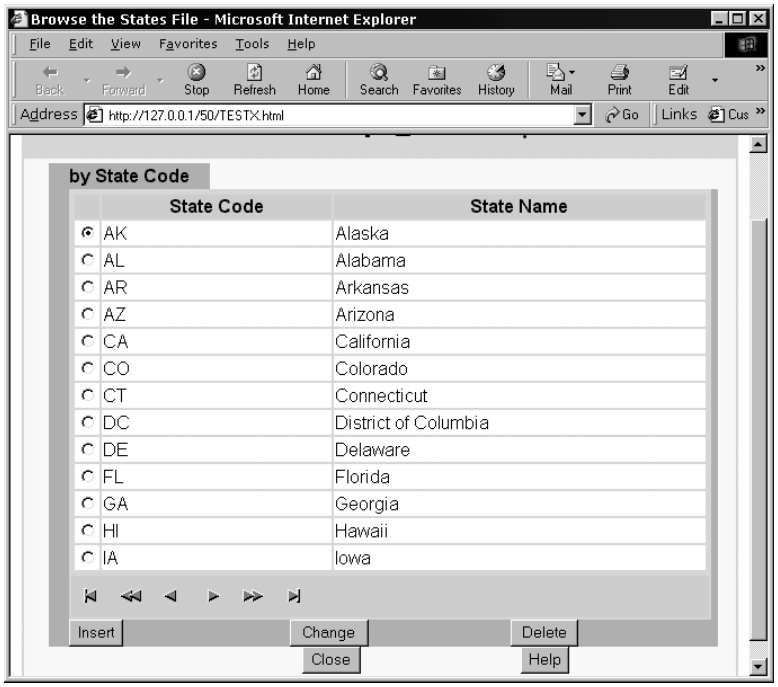
```
<TSSCRIPT tag=a attr=href replace="NAME" value="Name">
<a href="javascript:icSubmit('NAME$EventScrollTop');"><img ALT='First'
WIDTH="32" HEIGHT="32" SRC="PUBLIC/wizFirst.gif" border=0></a>
<a href="javascript:icSubmit('NAME$EventPageUp');"><img ALT='Prior'
WIDTH="32" HEIGHT="32" SRC="PUBLIC/wizPgUp.gif" border=0></a>
<a href="javascript:icSubmit('NAME$EventScrollUp');"><img ALT='Up'
WIDTH="32" HEIGHT="32" SRC="PUBLIC/wizUp.gif" border=0></a>
<a href="javascript:icSubmit('NAME$EventScrollDown');"><img ALT='Down'
WIDTH="32" HEIGHT="32" SRC="PUBLIC/wizDown.gif" border=0></a>
<a href="javascript:icSubmit('NAME$EventPageDown');"><img ALT='Next'
WIDTH="32" HEIGHT="32" SRC="PUBLIC/wizPgDn.gif" border=0></a>
<a href="javascript:icSubmit('NAME$EventScrollBottom');"><img ALT='Last'
WIDTH="32" HEIGHT="32" SRC="PUBLIC/wizLast.gif" border=0></a>
</TSSCRIPT>
```

The skeletons generate the following HTML code at runtime:

```
<!-- Table.htm -- Start -->
<table border="0" width="100%" bgcolor='#dcdcdc'>
 <tr>
  <td>
   <table border="0" width="100%">
   <tr bgcolor='#ccccff'>
   <th width="2">
     &nbsp;
   </th>
   <th>
     State Code
```

```
    </th>
    <th>
      State Name
    </th>
   </tr>
  <tr bgcolor='#ffffff'>
   <td width="2">
   <input type="radio" value='1' name='BROWSE_1$Choice' id='BROWSE_1$1'
checked=1>
   </td>
   <td>
     <LABEL FOR='BROWSE_1$1'>
     AK
    </LABEL>
   </td>
   <td>
    <LABEL FOR='BROWSE_1$1'>
     Alaska
    </LABEL>
  <!-- other rows and end tags edited for readability -->
<td bgcolor='#ccccff'>
<a href="javascript:icSubmit('BROWSE_1$EventScrollTop');"><img
ALT='First' WIDTH="32" HEIGHT="32" SRC='/wizFirst.gif' border=0></a>
<a href="javascript:icSubmit('BROWSE_1$EventPageUp');"><img ALT='Prior'
WIDTH="32" HEIGHT="32" SRC='/wizPgUp.gif' border=0></a>
<a href="javascript:icSubmit('BROWSE_1$EventScrollUp');"><img ALT='Up'
WIDTH="32" HEIGHT="32" SRC='/wizUp.gif' border=0></a>
<a href="javascript:icSubmit('BROWSE_1$EventScrollDown');"><img
ALT='Down' WIDTH="32" HEIGHT="32" SRC='/wizDown.gif' border=0></a>
<a href="javascript:icSubmit('BROWSE_1$EventPageDown');"><img ALT='Next'
WIDTH="32" HEIGHT="32" SRC='/wizPgDn.gif' border=0></a>
<a href="javascript:icSubmit('BROWSE_1$EventScrollBottom');"><img
ALT='Last' WIDTH="32" HEIGHT="32" SRC='/wizLast.gif' border=0></a>
<!-- Table.htm -- End -->
```

You will have a list box similar to this at runtime:


Browse the States File - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://127.0.0.1/50/TESTX.html
Go Links Cus
by State Code
Insert
Change
Delete
Close
Help

	State Code	State Name
●	AK	Alaska
○	AL	Alabama
○	AR	Arkansas
○	AZ	Arizona
○	CA	California
○	CO	Colorado
○	CT	Connecticut
○	DC	District of Columbia
○	DE	Delaware
○	FL	Florida
○	GA	Georgia
○	HI	Hawaii
○	IA	Iowa

Text.htm

This is the skeleton that generates the HTML version of a TEXT control.

The skeleton can get the attributes of the text control, and give you an HTML versions of the control.

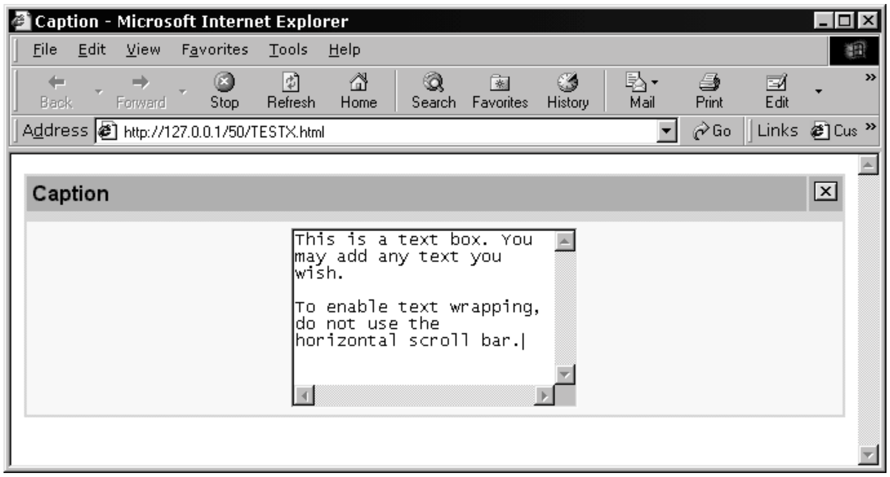
```
<!-- Text.htm -- Start -->
<textarea rows='9' cols='25' wrap=off name='ORD_ORDERNOTE'></textarea>
<!-- Text.htm -- End -->
```

The above is generated by the following skeleton code:

```
<!-- Text.htm -- Start -->
<TSSCRIPT value="EmbedBeforeControl" type=html></TSSCRIPT>
<TSSCRIPT tag=textarea attr=name value="Name">
<TSSCRIPT tag=textarea attr=disabled value=1 when="Disabled">
<TSSCRIPT tag=textarea attr=readonly value=1 when="Readonly">
<TSSCRIPT tag=textarea attr=rows value="(Height+4)/8">
<TSSCRIPT tag=textarea attr=cols value="(Width+2)/4">
<TSSCRIPT tag=textarea attr=onChange text="icSubmitForm()"
when="SubmitOnChange">
<TSSCRIPT tag=textarea attr=wrap text="soft" when="!HScroll">
```

```
<textarea rows="1" cols="20" wrap=off>
<TSSCRIPT value=DisplayText>String Text</TSSCRIPT></textarea>
<TSSCRIPT value="EmbedAfterControl" type=html></TSSCRIPT>
<!-- Text.htm -- End -->
```

The text control could look similar to this:


Caption - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Forward
Stop
Refresh
Home
Search
Favorites
History
Mail
Print
Edit
Address http://127.0.0.1/50/TESTX.html
Go
Links
Cus
Caption

```
This is a text box. You
may add any text you
wish.

To enable text wrapping,
do not use the
horizontal scroll bar.
```

Toolbar.htm

All this skeleton code does in define an area in which button controls are placed. See Button skeleton.

Summary

If you understand the pieces of the skeletons, then you can see how they fit together. You can even author your own skeletons and simply write TSSCRIPT commands to include them.

6 - Common Questions and Answers

Introduction

This section covers several common questions that we found to be helpful with getting your application running quickly. The focus of these questions are the Skeletons, although it is not restricted to them. For more information on Skeletons, see the *Skeleton* chapter in this manual. The purpose of this section is to offer as many real world issues as possible.

When an application is run in a browser, you can see the HTML code generated by the skeletons (RIGHT-CLICK on a blank area of a page and choose **View Source**). This is the best way to understand how the skeletons interact with your program to produce the final result.

Common Questions

How do I set background colors for pages in my application?

This is controlled from the *Window.htm* skeleton. In this file, you will see a TSSCRIPT line that is a comment about colors. The next set of lines defines TSSCRIPT tags and their attributes. One of the attributes is a *default* color. Lets examine one of these lines:

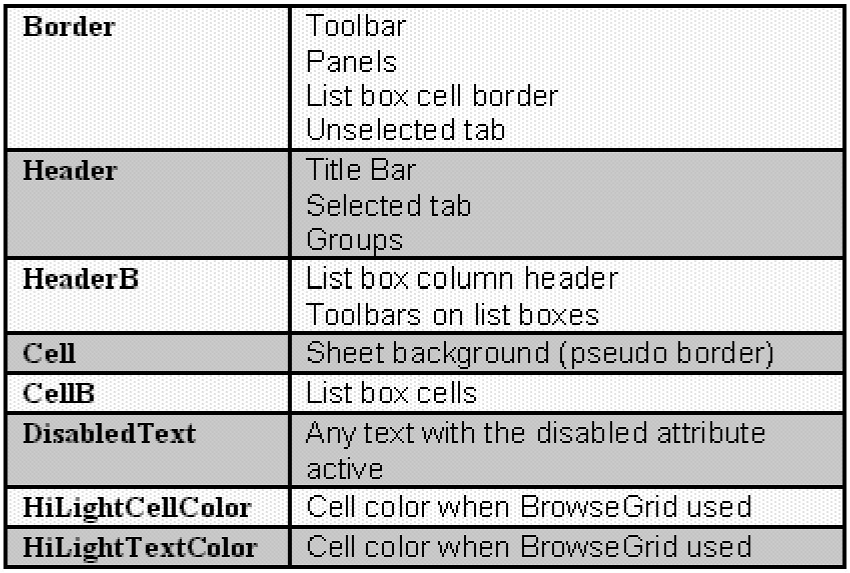
```
<TSSCRIPT tag="<* FinalColor=CellB>" attr=bgcolor value="'#ffffff'"
comment="Cell background color" phase=*>
```

What this line is doing is stating that a new FinalColor tag, named CellB, has a default background color attribute with value *#ffffff*. The *phase*=* means that is can be overridden by any phase value later, for example; *runtime*. If you examine a few lines further down you see this line:

```
<TSSCRIPT tag="<* FinalColor=*>" attr=FinalColor remove phase=Runtime
comment="Remove pseudo tags from the table entries">
```

This line is simply stating that whatever tag is being used now, use whatever color is setup at runtime. This tells you that the line you need to change is the first one. You can replace the #ffffff with another color, for example, ‘Green’.

The following is a list of color attributes and what controls they affect.



Border	Toolbar Panels List box cell border Unselected tab
Header	Title Bar Selected tab Groups
HeaderB	List box column header Toolbars on list boxes
Cell	Sheet background (pseudo border)
CellB	List box cells
DisabledText	Any text with the disabled attribute active
HiLightCellColor	Cell color when BrowseGrid used
HiLightTextColor	Cell color when BrowseGrid used

How can I set a default font?

You can do this with the <FONT> tag. Since there are skeletons for each type of control (CHECK, STRING, TAB, etc), setting the font for each of these files is labor intensive. It also is considered a “no-no” for HTML 4.x specifications. See http://www.w3.org/MarkUp/ for comments about this. If you know you will need to support older browsers, then use the <FONT> tag. The previous link has specifics about this.

How can I implement Cascading Style Sheets?

A better way of using fonts is *Cascading Style Sheets (CSS)*. For a good reference on CSS standards, see http://www.w3.org/Style/CSS/ for a list of many resources on this subject.

In short, a CSS sets a style for fonts and appearance and is used on tags, for example <p> which begins a new paragraph. You could think of these as the event embeds for HTML. When a new paragraph happens, insert the new CSS and activate it.

Note:

Not all browsers support this relatively new feature. The above link has a list of browsers (and minimum version) that do.

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>This displays text in the title bar</title>
<link rel="stylesheet" type="text/css" href="../mysheets/kewlstyle.css">
</head>
```

Skeletons support this feature too. You could modify one to use them. Here is an example:

```
<!-- Window.htm -- Start -->
<html>
<head>
<title>
The Best of FAQs</title>
<style type="text/css">
</style>
<style type="text/css">
</style>
```

You could also change the window skeleton as follows:

```
<TSSCRIPT value=”EmbedBeforeHeadClose” type=html></TSSCRIPT>
<style type=”text/css”><!–td{font-family: verdana,arial,sans serif;font-
size: x-small;}–></style>
<style type=”text/css”><!–pre{font-family: courier new,courier;font-size:
x-small;}–></style>
</head>
```

How can I have an image with text on a button?

Here is a way to make an image a link if your Skeleton (Image.htm) has the line:

```
<a><img></a>
```

to display the image.

Bracket this with TSSCRIPT as shown below.

```
<TSSCRIPT tag=a attr=href value="ImageLink" when="ImageLink!=''">
<a><img></a>
</TSSCRIPT>
```

Add the image in the IDE. On the Position tab set the width and height of the button. In Internet Options - Controls - Properties - Properties - Insert.

Name: *ImageLink*

Type: *String*

Value: *'http://127.0.0.1/default.htm'*

Do not check the *Refresh when changed* on the Events Tab.

Save all dialogs and compile and run.

The HTML that is sent to the client looks like:

```
<!-- Image.htm -- Start -->
<a href='http://127.0.0.1/default.htm'><img src='/public/MyImage.jpg'
width=88 height=21></a>
<!-- Image.htm -- End -->
```

As an alternative, you can make HTML buttons with images. This is done with buttons. For example:

```
<BUTTON name=”submit” value=”submit” type=”submit”> Send<IMG
src=”/image.gif” alt=”text on the button”></BUTTON>
```

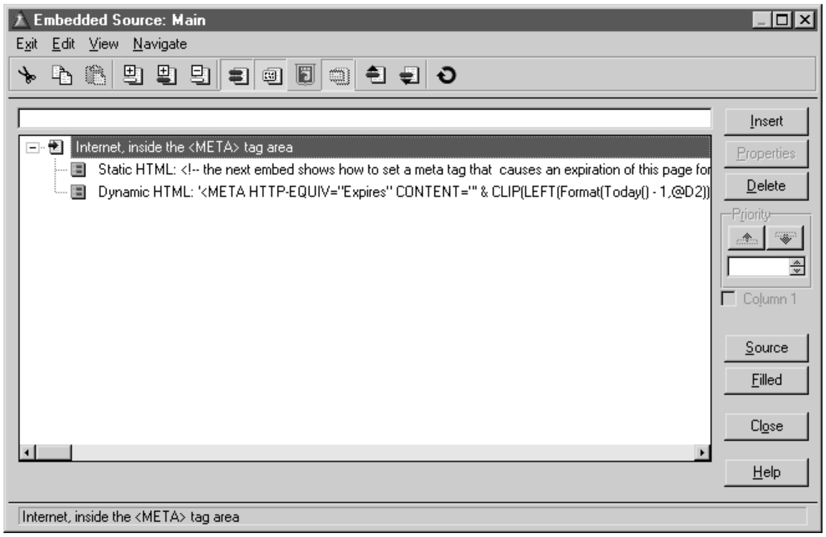
In this example, the button is used to send data on a <FORM>.

How can I get better control over size & placement of controls?

Place related groups of controls inside Group structures on windows. Tables that are generated are generated around these structures instead of around the individual controls.

How can I use meta-tags?

To use meta-tags in a Clarion application, go to the embeds for a procedure. Find the embed point, *Internet, inside the <META> tag area*. Insert your Dynamic or Static HTML here.



Internet, inside the <META> tag area
Static HTML: <!-- the next embed shows how to set a meta tag that causes an expiration of this page for
Dynamic HTML: '<META HTTP-EQUIV="Expires" CONTENT="' & CLIP(LEFT(Format(Today() - 1,@D2))

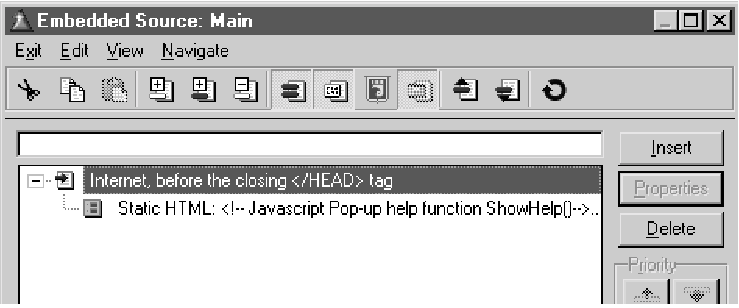
Internet, inside the <META> tag area

How can I make a pop-up window for data validation?

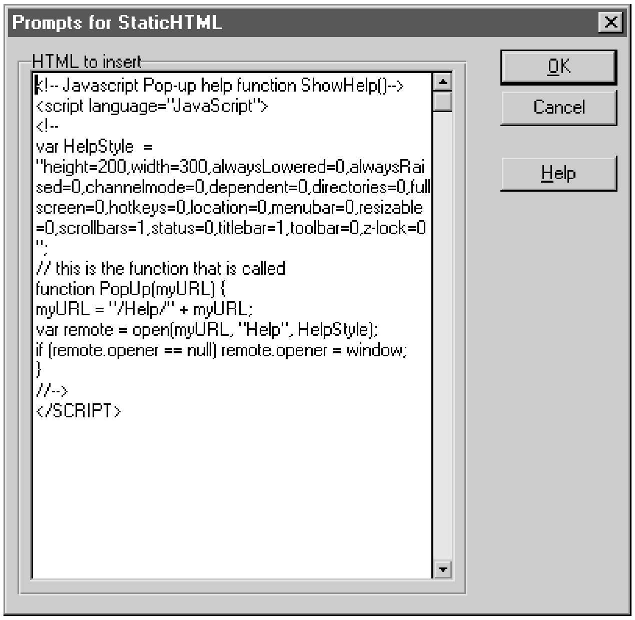
A commonly user web page technique is to open a new window when a link is selected.

```
<a
HREF=”JavaScript:void(0)”onClick=”window.open(‘http://www.softvelocity.com/cws/
c5launch.dll/example/example.exe.0’,
’’,’toolbar=no,directories=no,captionbar=no,status=yes,menubar=no,scrollbars=ye
s,
location=no,width=550,height=400,resizable=yes’);”
onmouseover=”self.status=’Just the FAQs’; return true “>Go therenow</a>
```

The embedded HTML code should be added in the *Internet, before closing </HEAD> tag* embed point.



Internet, before the closing </HEAD> tag
Static HTML: <!-- Javascript Pop-up help function ShowHelp()-->..



```
<!-- Javascript Pop-up help function ShowHelp()-->
<script language="JavaScript">
<!--
var HelpStyle  =
"height=200,width=300,alwaysLowered=0,alwaysRai
sed=0,channelmode=0,dependent=0,directories=0,full
screen=0,hotkeys=0,location=0,menubar=0,resizable
=0,scrollbars=1,status=0,titlebar=1,toolbar=0,z-lock=0
";
// this is the function that is called
function PopUp(myURL) {
myURL = "/Help/" + myURL;
var remote = open(myURL, "Help", HelpStyle);
if (remote.opener == null) remote.opener = window;
}
//-->
</SCRIPT>
```

What is the difference between POST and GET and how do I change between the two?

GET and POST are two ways that information is passed to the server from an application. By default, a Clarion application uses the GET method.

The GET communicates with the server by appending the form data to the URL specified by the action attribute (with a question-mark (“?”) as separator) and this new URL is sent to the processing agent.

The POST method, communicates with the server by including the data in the body of the form. It is then sent to the processing agent.

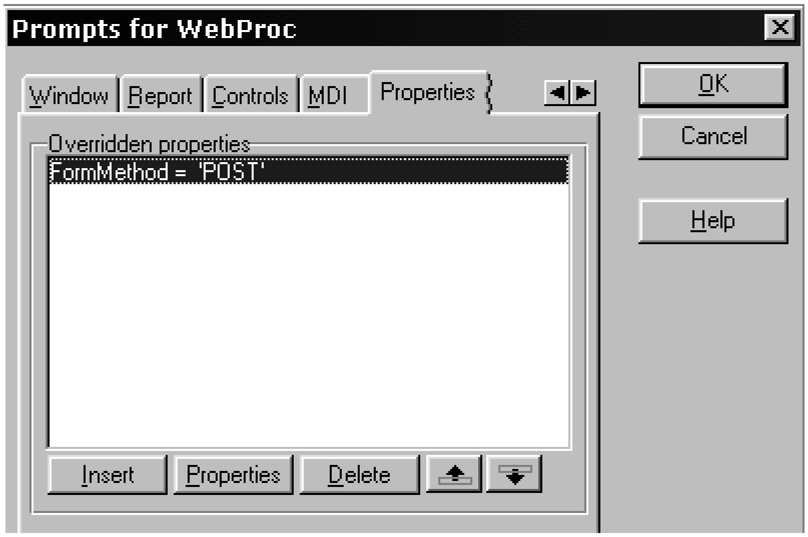
The GET method should be used when the form does not make changes to a database or side-effects. Many database searches have no visible side-effects and make ideal applications for the GET method.

If the service associated with the processing of a form causes side effects (for example, if the form modifies a database or subscription to a service), the POST method should be used.

Note:

The GET method restricts form data set values to ASCII characters. Only the POST method (with enctype=”multipart/form-data”) is specified to cover the entire [ISO10646] character set.

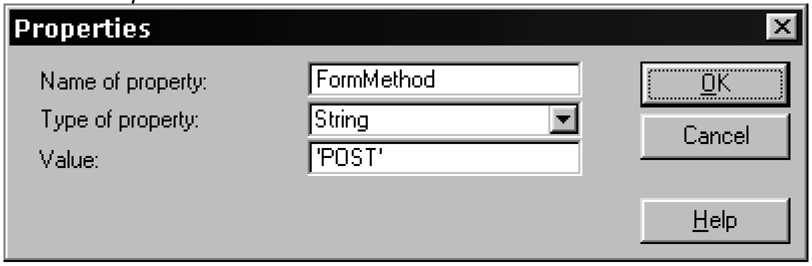
To set this, you should be on the extensions for the procedure you wish to change.



Overridden properties

FormMethod = 'POST'

On the *Properties* tab enter:



Name of property: FormMethod

Type of property: String

Value: 'POST'

The Window.htm skeleton has these lines in it (partially shown):

```
<TSSCRIPT tag=form attr=method value="FormMethod">
<TSSCRIPT tag=form attr=enctype value="FormEncoding">
  <form name="ClarionForm" method="GET"
```

The above defines a tag called *form* with a value of *Form Method*. The form name is the name of this tag and its method. The template dialog simply changes the default value of GET to POST.

If you wish to change this globally, then edit *window.htm*.

How can I get server variables and their values?

You can obtain whatever information about properties that you wish from a skeleton.

What you need to do is to specify a Skeleton Property and then derive the GetProperty method and return the global variable instead.

For example : Glo:Amount

In the skeleton:

```
<TSSCRIPT value="Amount"></TSSCRIPT>
```

In the procedure:

```
GetProperty
  IF name="Amount"
      RETURN CreateStringValue(Glo:Amount)
```

There is also a code template available to accomplish this task.

How can I create tooltips?

To have your buttons or images display a tooltip, place the text in the Tip prompt on the control properties Help tab. The TIP will become an ALT= HTML attribute of the control.

How can I launch a Clarion application from a link?

You need to provide a link on the web page where the Clarion application is to be called from. This can be an image, hyperlink, button, etc, depending on the effect you wish. Here is an example:

```
<a href="http://somesite.com/cws/c5launch.dll/demos/demo.exe.0" Click
here for a demo.</font></a>
```

How can I add email capability to my applications?

All you need is an entry control for the email address. However, this does not give you email capability. For the entry control, that contains the address, tell the skeletons about the extended capability for the entry control. This is done via the Web procedure extension. Just add *email* in the Extra Capabilities entry.

This skeleton formats data from a variable containing an email address so it is a “Mailto:” hyperlink. To use this skeleton, you would specify the *email* capability property in the **Capabilities** prompt in the Internet Connect template in the IDE (individual overrides for a control).

```
<HTML>
<head>
<meta name=”ts-control” content=”sstring”>
<meta name=”ts-capabilities” content=”email”>
</head>
<BODY>
<!— email.string.htm — Start —>
<TSSCRIPT value=”EmbedBeforeControl” type=html>
</TSSCRIPT>
<TSSCRIPT tag=a attr=href replace=NAME value=Contents>
<A HREF=”mailto:NAME”>
<TSSCRIPT value=Contents>
</TSSCRIPT>
</A>
</TSSCRIPT>
<TSSCRIPT value=”EmbedAfterControl” type=html>
</TSSCRIPT>
<!— email.string.htm — End —>
</BODY>
</HTML>
```

Part II

Internet Connect

7 - Tutorial—Making a Web Application

In Clarion, you can create an application from a data dictionary—with no coding required. All you need to do is create the Data Dictionary then use the Application Wizard to make a complete Windows application—in minutes! With Internet Connect, the Application Wizard has an additional checkbox that lets you Web-enable the application you are creating. This allows you to create a Web application with only one additional click of your mouse!

In this chapter, you will:

- Use the **Application Wizard** to create a hybrid Web/Windows application from a Clarion Data Dictionary, then run the program using your browser.
- Compile and deploy the application, then run it in a browser.
- Optimize that application for the Web using the template interface, recompile, deploy it, and run it again.
- Modify the appearance of the application for the Web, recompile, deploy it, and run it again.

This should all take about thirty minutes—without any “coding” on your part. By the end of this chapter, you’ll have a complete application for a simple order entry system.

Let’s get started!

Web Application Wizard

Creating a Hybrid Web/Windows Application

Starting Point:
You should have the Clarion development environment open.

This tutorial assumes that you installed Clarion in C:\Clarion6 and the Application broker in C:\CWICWEB. If you used different directories, you will have to modify the instructions accordingly. This tutorial also assumes that you have completed the tutorials in the Clarion *Getting Started* and *Learning Clarion* topics and have a basic familiarity with the Clarion development environment.

Create your first Clarion Web application

1. On the **Pick** dialog, select the *Application* tab, then press the **New...** button.

   This opens the *New* dialog.

2. **Navigate** to the *\Clarion6\Examples\WebTutor* folder from the Directories list.

3. Type *WebOrder* in the **File Name** field, then press the **Save** button.

   This opens the *Application Properties* dialog.

4. Press the elipsis (...) button to the right of the **Dictionary File** entry box.

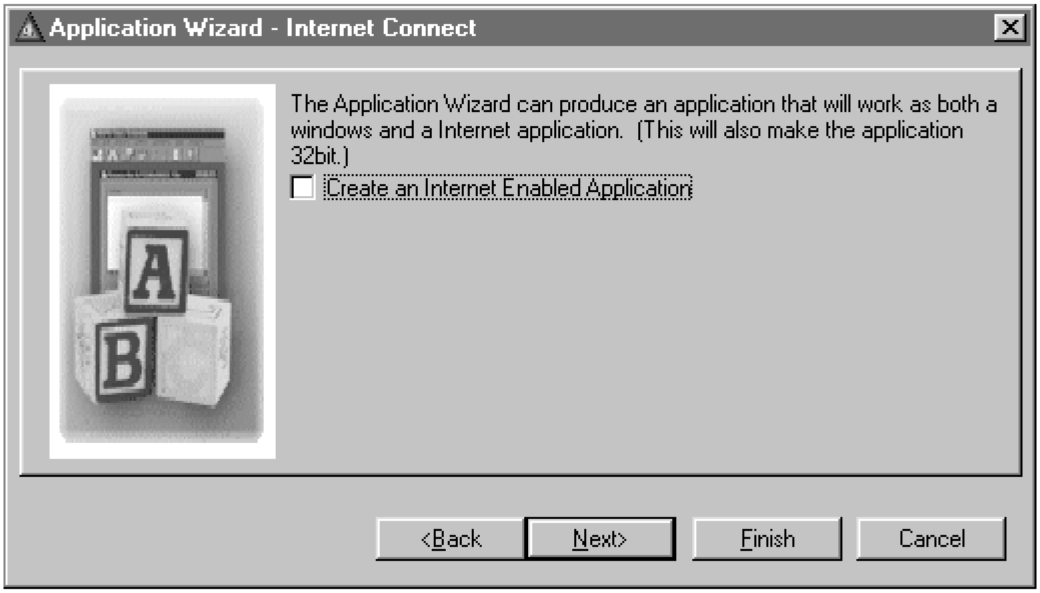
   This opens the *Select Dictionary* dialog.

5. Highlight the *WebOrder.dct* (in the C:\Clarion6\Examples\WebTutor\ directory) file then press the **Open** button.

Run the Application Wizard

1. Check the **Application Wizard** box, then press the **OK** button. If the *Select Application Wizard* window appears, highlight *Application Wizard – Create a New Database Application*, and then press the **Select** button.

2. After the Application Wizard opens, press the **Next** button past the next five wizard screens, accepting the defaults. Make sure that the **Generate Procedures for all files in my dictionary** check box is checked, and that the **Control Model** is set to *Toolbar*.

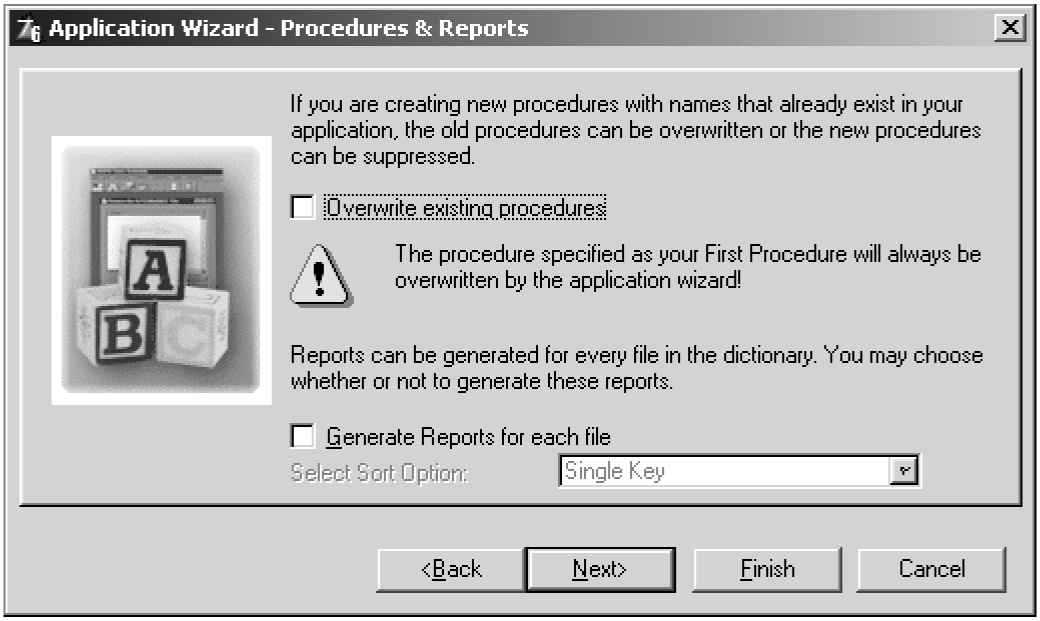


Application Wizard - Internet Connect

The Application Wizard can produce an application that will work as both a windows and a Internet application. (This will also make the application 32bit.)

☐ Create an Internet Enabled Application

3. ***Do Not*** Check the **Create an Internet Enabled Application** box, then press the **Next** button. This step adds the Web Builder extension template to the application. In this tutorial, we will be using the Internet Deployment template set.



Application Wizard - Procedures & Reports

If you are creating new procedures with names that already exist in your application, the old procedures can be overwritten or the new procedures can be suppressed.

☐ Overwrite existing procedures

The procedure specified as your First Procedure will always be overwritten by the application wizard!

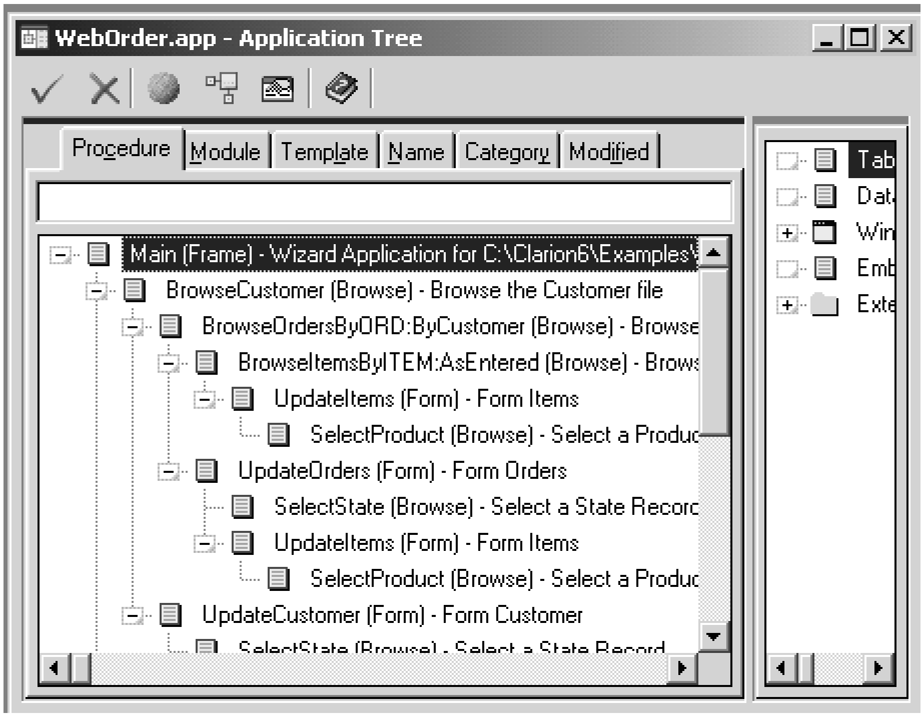
Reports can be generated for every file in the dictionary. You may choose whether or not to generate these reports.

☐ Generate Reports for each file

Select Sort Option: Single Key

4. Uncheck the **Generate Reports for each file** box, then press the **Finish** button.

The Application Wizard creates the application.

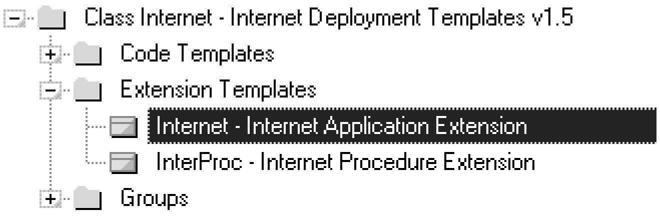


- Main (Frame) - Wizard Application for C:\Clarion6\Examples\
  - BrowseCustomer (Browse) - Browse the Customer file
    - BrowseOrdersByORD:ByCustomer (Browse) - Browse
      - BrowseItemsByITEM:AsEntered (Browse) - Brows
        - UpdateItems (Form) - Form Items
          - SelectProduct (Browse) - Select a Produc
      - UpdateOrders (Form) - Form Orders
        - SelectState (Browse) - Select a State Recorc
        - UpdateItems (Form) - Form Items
          - SelectProduct (Browse) - Select a Produc
    - UpdateCustomer (Form) - Form Customer
      - SelectState (Browse) - Select a State Record

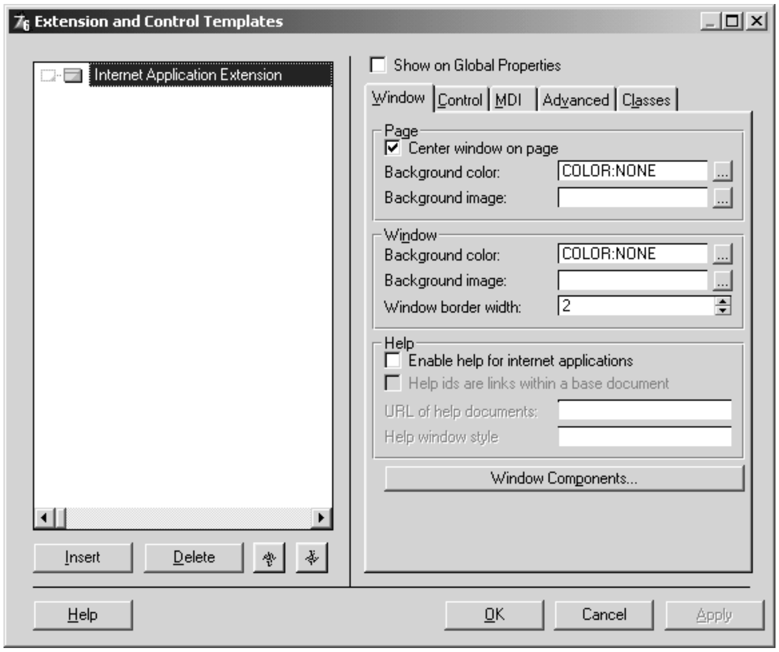
Add the Internet Deployment Templates

1. Press the **Global** Icon Button to access the *Global Properties* window.
2. In the *Global Properties* window, press the **Extensions** button.

3. Press the **Insert** button to add the *Internet Deployment Application Extension* as shown below:



Class Internet - Internet Deployment Templates v1.5
  Code Templates
  Extension Templates
    Internet - Internet Application Extension
    InterProc - Internet Procedure Extension
  Groups



☐ Show on Global Properties

Page
☑ Center window on page
Background color: COLOR:NONE
Background image:

Window
Background color: COLOR:NONE
Background image:
Window border width: 2

Help
☐ Enable help for internet applications
☐ Help ids are links within a base document
URL of help documents:
Help window style

4. Press the **OK** button to close the *Extension and Control Template* window. Press the **OK** button on the *Global Properties* window to return to the Application Tree.

Make the Application

1. Choose Project ▸ Make (or press the Make icon button on the toolbar).

   Congratulations! Your first Web application is ready to deploy and run.

2. Press the **OK** button on the compile results window.

Deploying the Application

The last step created *WebOrder.exe*. Since it is a Web-enabled application, it can now run under Windows as a standard Windows executable or over the Web through the Application Broker using a browser. Next we will deploy the application and the files it needs to execute. Note that we are deploying this to a different directory on the same machine, but the process would be the same to deploy the program to a server machine.

1. Open Windows Explorer (or Windows NT Explorer).

2. Copy *WebOrder.exe* from the *C:\Clarion6\Examples\Webtutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory (You may have to create this directory).

   Note:

   We have provided sample data files in both directories. If you had local data files, you would also need to deploy them.

3. Copy the files listed below from the *C:\Clarion6\BIN* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

   C60RUNx.DLL

   C60TPSx.DLL

   C60ASCx.DLL

   C60DOSx.DLL

   These are the support DLLs your application uses, including the runtime library and database drivers.

   This step is included here even though it may not be necessary on your WIN 95/98 development machine because these files are in your PATH. However, NT server and XP behaves differently. Each user has a PATH and deploying the DLLs with the .EXE ensures that the user accessing the application through a browser has the support files available. This is explained in detail in *Deploying Applications*.

4. Start the Application Broker by double-clicking on *C60APS10.exe* (or *C60APS.exe* if you have the full version of the Application Broker) in the *C:\CWICWEB\* directory.

5. Start your favorite browser.

   Next, test the Application Broker and your TCP/IP setup using the Localhost loopback method:

6. On the Browser's URL line, type:

   `http://localhost/btest.htm`

   or

   `http://127.0.0.1/btest.htm`

   then press ENTER.

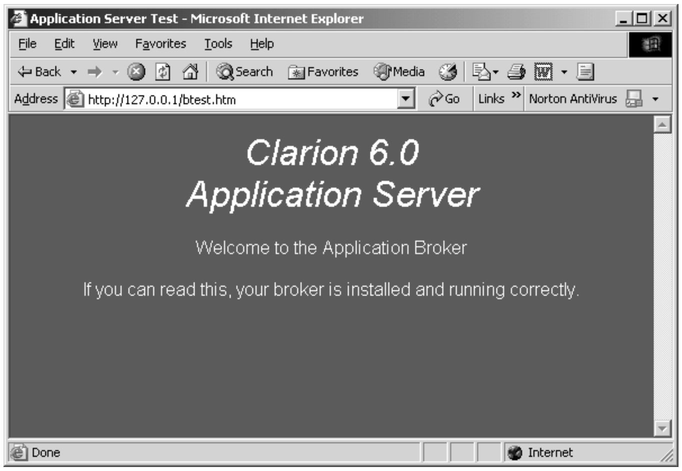
Note:

If you have the broker set to a port other that port 80, you must add that to the domain portion of the URL. For example:

`http://localhost:8080/btest.htm`

or

`http://127.0.0.1:8080/btest.htm`



Application Server Test - Microsoft Internet Explorer

Address: http://127.0.0.1/btest.htm

Clarion 6.0
Application Server

Welcome to the Application Broker

If you can read this, your broker is installed and running correctly.

Done

If the test Web page displays correctly, you have the application broker installed and running correctly. If not, you should return to the previous chapter and reconfigure your setup.

Next, start the application in the browser:

7. On the Browser's URL line, type:

   `http://localhost/exec/webtutor/weborder.exe.0`

   or

   `http://127.0.0.1/exec/webtutor/weborder.exe.0`,

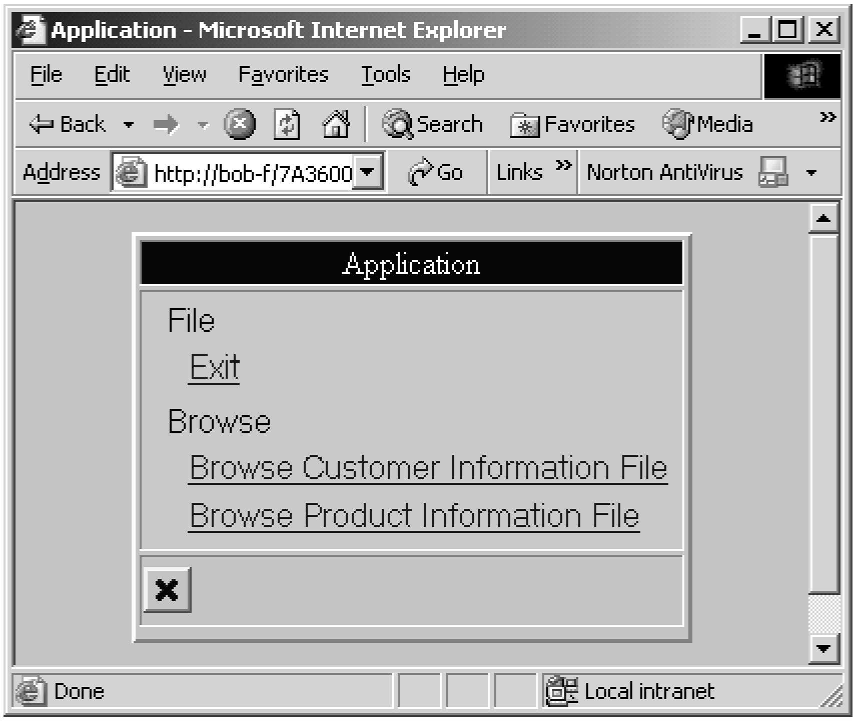
   then press ENTER.

Note:

If you have the broker set to a port other that port 80, you must add that to the domain portion of the URL. For example:

`http://localhost:8080/exec/WebTutor/WebOrder.exe.0`

Congratulations! Your first Web application is running.


Application - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://bob-f/7A3600 Go Links Norton AntiVirus
Application
File
Exit
Browse
Browse Customer Information File
Browse Product Information File
Done
Local intranet

Now you can explore this new application and compare it to the manner in which it runs under Windows. You will notice that there are some minor differences between the two, because of the platform, but it will look and feel very much the same.

8. When you are finished, click on the **Exit** hyperlink.

This closes the application. Notice the browser now displays a blue Web page with a hyperlink to restart the application. This page is created by the application broker automatically unless you specify a page to return to on exit in the Global Internet Application Extension template.

Leave your browser open with the restart page displayed. You will use this page to restart your application.

The rest of this chapter walks you through techniques for optimizing your application for the Web platform. This will not only demonstrate some features in the IBC templates, but will also show you how much power you have when you finally do write your own code to provide some “non-standard” functionality.

Continue on! You’ve only just skimmed the surface of Clarion Internet Connect, *and there’s a lot more*!

Faster is Better—Optimizing your Application

The Web introduces one additional programming challenge—bandwidth conservation. It is important to utilize all the methods available to reduce the amount of data transmitted over the network. Many users connect to the Web using a modem and telephone lines, which is a relatively slow network connection.

Internet Connect is Designed to Conserve Bandwidth

Clarion Internet Connect was designed to conserve bandwidth. The Java controls it creates most often update dynamically on the client browser without the need to refresh the entire page. This form of “dynamic HTML” requires only a small amount of data to be transmitted. This is known as a Partial Refresh. When a page is partially refreshed, only the controls which are enabled to accept updated data redisplay. Entry Controls, Java String controls, Java Image controls, and Java Listboxes are usually enabled to update dynamically.

For the same reason (bandwidth conservation) many controls trigger a Partial Refresh. For example, selecting a new record in a listbox triggers a Partial Refresh, allowing most controls to redisplay current data.

Partial Refresh versus Full Refresh

There are some instances, however, where a Partial Refresh is appropriate but is not the default. Changing events to trigger a Partial Refresh instead of a Full Refresh, where appropriate, is one of the best ways to optimize your Web applications.

There are many cases when a Partial Refresh is appropriate but a Full Refresh is the default. This is because the templates cannot anticipate every possibility and must favor the safer Full Refresh instead of the faster Partial Refresh.

For example, a multi-sorted list which has no additional controls populated on the Tabs performs better if you use Individual Control Overrides to specify a Partial Refresh when a new tab is selected. This will only change the data in the listbox instead of replacing the entire page.

Let’s look at the application we just created.

1. Task-switch back to your browser.
2. CLICK on the restart hyperlink.

   The WebOrder application appears inside the browser.
3. CLICK on the **Browse Customer Information File** hyperlink.

The Browse the Customer File “window” appears in the browser. Notice that the window contains a listbox and two tabs. Clicking on a tab changes the sort order of the list.

4. CLICK on each of the tabs and notice the behavior of the Web page.

   You should have noticed that the entire page was replaced to redisplay the list. This is the default behavior of sheets and tabs. In the next section we will override this default behavior.

5. CLICK on the blue X button at the right end of the toolbar to close the Browse window, then click on the *Exit* hyperlink to exit the application.

   Leave your browser open with the restart page displayed. You will use this to restart your application after making some changes.

Internet Procedure Extension Template

In this section, we will override the SHEET control’s default action to optimize it for performance over the Web.

Starting Point:
You should have the weborder.app open in the Clarion development environment .

1. In the Application Tree, select the **Category** tab.

   This sorts the procedures by category. Notice there are seven procedures within the *Browse* category.

2. Highlight the *BrowseCustomer* procedure, then press the **Properties** button.

   This opens the *Procedure properties* window.

3. Press the **Internet Options** button.

4. Select the **Controls** Tab.

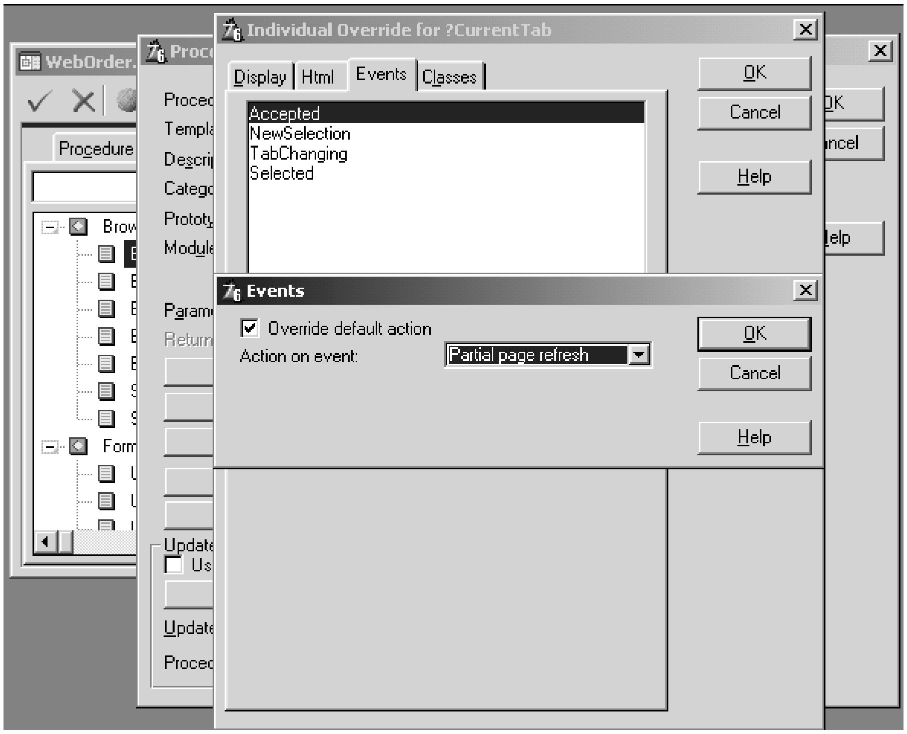
5. Highlight the Sheet control (*?CurrentTab*) in the **Individual Control Options** list.

6. Press the **Properties** button, then select the **Events** tab.

7. Highlight the *Accepted* event, then press the **Properties** button.

Override the Default Full Refresh with Partial Refresh

1. Check the **Override default action** box, then select *Partial page refresh* from the drop-down list.



Accepted
NewSelection
TabChanging
Selected

Events

☑ Override default action

Action on event: Partial page refresh

2. Press the **OK** buttons on all the windows until you return to the application tree (4 times).

3. Repeat these steps for all other Browse procedures.

4. Choose **Project** ▸ **Make** (or press the Make icon button on the toolbar).

   Your Web application is ready to deploy once again.

5. Open Windows Explorer (or Windows NT Explorer).

6. Copy *Weborder.exe* from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

   This time you need only deploy the application, the DLLs have not changed.

Let’s run the application to see how the changes we made affect its behavior.

See the difference

1. Task-switch back to your browser.
2. Start the application in the browser by clicking on the Restart hyperlink.
3. CLICK on the **Browse Customer Information File** hyperlink.
4. CLICK on each of the tabs and notice the behavior of the Web page.

   You should notice that the list now re-displays data without sending an entire page.
5. Exit the application.

   Leave your browser open with the restart page displayed. You will use this to restart your application after making some changes.

Looks are Important—Adding Graphics

The Web has produced a colorful, enjoyable medium for computer users. Many Web sites are designed to provide both content and an attractive interface. Clarion Internet Connect has support for the most commonly used methods of employing graphics and colors in Web pages.

In this section we will add a background image to the pages in which the application's windows appear. This provides a back-drop for the running program and helps to visually indicate the portion that is the application and the portion that is not.

This section of the tutorial is not intended to teach you page design or artistic methods. Ths section is designed to show you how to use the template interface to create the look-and-feel you want.

Internet Application Extension Template

First, we will add a background image:

Starting Point:
You should have the weborder.app open in the Clarion development environment.

1. In the Application Tree, press the **Global** button. This opens the *Global Properties* window.

2. Press the **Extensions** button.This opens the *Extensions and Control Templates* window.

3. Highlight *Internet Application Extension*.

4. In the Page area, press the ellipsis (...) button next to Background Image. This opens the standard Windows file dialog.

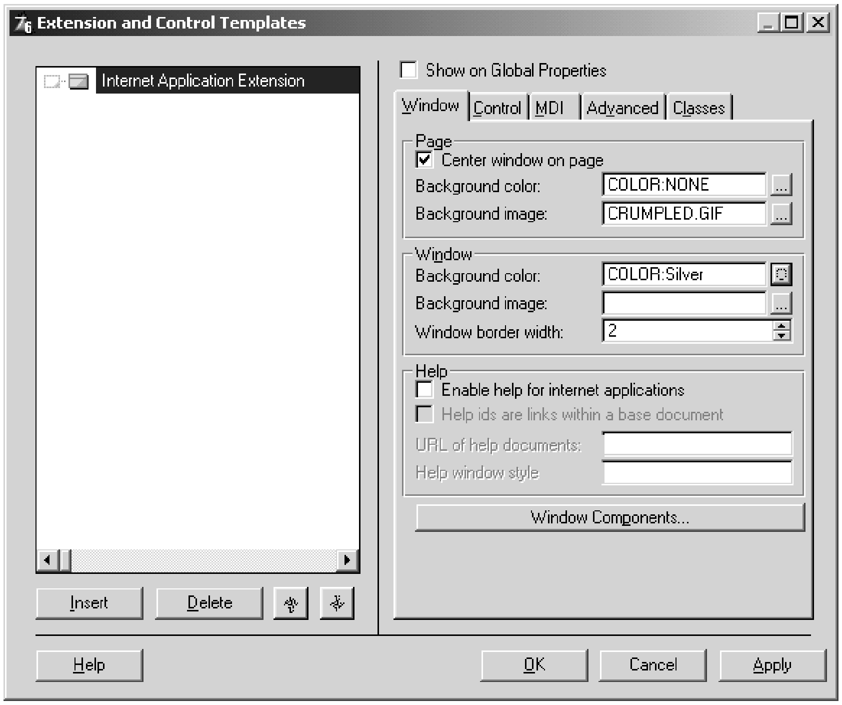
5. Select *Crumpled.gif*, then press the **OK** button..

   This adds a tiled image to the Web page background. The image is of a crumpled piece of grey paper. Keep in mind that this image file will need to be deployed.

6. In the Window area, press the ellipsis (...) button next to **Background Color**. This opens the standard Windows color dialog.

7. Select the *Silver* color, then press the **OK** button.

   This adds a background color attribute to the HTML representation of the application's window. In addition to adding the color, this also prevents the background image from showing through.



☐ Show on Global Properties

Page
☑ Center window on page
Background color: COLOR:NONE
Background image: CRUMPLED.GIF

Window
Background color: COLOR:Silver
Background image:
Window border width: 2

Help
☐ Enable help for internet applications
☐ Help ids are links within a base document
URL of help documents:
Help window style

8. Press the **OK** button on the *Extensions and Control Templates* and the *Global Properties* window.

Make, Deploy, and Run the Application

1. Choose **Project** ▸ **Make** (or press the Make icon button on the toolbar).

   Your Web application is ready to deploy once again.

2. Open Windows Explorer (or Windows NT Explorer).

3. Copy *Weborder.exe* from the *C:\Clarion6\Examples\WebTutor* directory to the C:\CWICWEB\EXEC\WebTutor directory.

4. Copy *Crumpled.gif* from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\Public* directory.

5. Task-switch to your browser and restart the application. Notice the new look.

In this chapter we learned how to make a new application and make some basic changes to optimize it for performance and appearance when running over the Web. In the next chapter, we will Web-enable an existing application, so you can learn to publish any of your applications on the Web.

8 - Tutorial— Web-Enabling an Existing Application

Porting an existing Clarion application to the Web is just as easy as creating a new Web application.

In this chapter we will use WebTree.APP.

In this chapter, you will:

- Use the IBC templates to port an existing Clarion application to the Web.
- Compile and deploy the application, then run it in a browser.
- Learn about using Tree controls on the Web and deploying icons.
- Optimize the Tree display using techniques similar to those used in the first tutorial.

This should all take about fifteen minutes. By the end of this chapter, you’ll have a complete application for a simple order entry system using a different interface than the application used in the first tutorial.

Let’s get started!

Using the Global Internet Application Extension Template

Porting an Application to the Web

Starting Point:
You should have the Clarion development environment open.

This tutorial assumes that you installed Clarion in C:\Clarion6 and the Application broker in C:\CWICWEB. If you used a different directory, you will have to modify the instructions accordingly.

Web-enabling a Clarion application

1. From the *Pick* dialog, press the **Open...** button. This opens the *Open* dialog.
2. Select the **Application** tab.
3. Select the *C:\Clarion6\Examples\WebTutor* directory from the Directories list, select *WebTree.app*, then press the **Open** button. This opens the *Application Tree* dialog.
4. In the Application Tree, press the **Global** icon button. This opens the *Global Properties* window.
5. Press the **Extensions** button. This opens the *Extensions and Control Templates* window.
6. Press the **Insert** button.
7. Highlight *Internet Application Extension*, then press the **Select** button.

   This adds the *Internet Application Extension* template which automatically adds the *Internet Procedure Extension* template to each procedure in the application.
8. Press the **OK** button on the *Extensions and Control Templates* and the *Global Properties* windows.

That’s all it takes to Web-enable an existing application!

Make and Deploy

1. Choose **Project** ▸ **Make** (or press the Make icon button on the toolbar).

   Your Web application is ready to deploy.
2. Press the **OK** button on the compile results window.

3. Open Windows Explorer (or Windows NT Explorer).

4. Copy *WebTree.exe* from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

5. Copy all the icon files (*.ICO) from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\Public* directory.

   These icons are used on the Toolbar buttons and in the Tree control. They must be deployed to the \PUBLIC directory in order for the browser to display them. The icons in the Standard toolbar which the earlier tutorial application used are compiled into the Java classes and need not be deployed.

Run the application

1. Start the Application Broker by double-clicking on *C60APS10.exe* (or *C60APS.exe* if you have the full version of the Application Broker) in the C:\CWICWEB\ directory.

Note:

As in the first tutorial, we will use the executable version of the Application Broker. The ISAPI version works in a similar manner, with a only few differences. These are discussed in the Application Broker chapter.

2. Start your browser.

3. Next, start the WebTree.exe application in the browser. (http://localhost/exec/webtutor/webtree.exe.0)

Examine the application

You should notice that this application looks a little different than the previous application. It uses a toolbar but no menu. This is a common interface in Web applications, so this technique bears demonstration here.

1. CLICK on the **Orders** button.

   The *Browse Customer Orders* “window” appears in the browser. Notice that the window contains a Tree control and two buttons to **Expand All** and **Contract All**.

2. CLICK on the **Expand All** and **Contract All** buttons and notice the behavior.

   Notice that expanding and contracting the tree refreshes the entire page. We will use the same partial refresh technique you learned in the first tutorial to optimize this behavior.

3. Exit the application (by pressing the blue X).

Leave your browser open with the restart page displayed. You will use this to restart your application after making some changes.

Overriding the default action

In this section, we will override the BUTTON control's default action to optimize it for performance over the Web.

Starting Point:
You should have the WebTree.APP open in the Clarion development environment.

1. Highlight the *BrowseCustomer* procedure, then press the **Properties** icon button.

   This opens the *Procedure properties* window.

2. Press the **Internet Options** button.
3. Select the **Controls** Tab.
4. Highlight the Button control (*?Expand*) in the **Individual Control Options** list.
5. Press the **Properties** button, then select the **Events** tab.
6. Highlight the *Accepted* event, then press the **Properties** button.
7. Check the **Override default action** box, then select *Partial page refresh* from the drop-down list.
8. Press the **OK** buttons on the *Events* and *Individual Overrides* windows.
9. Highlight the Button control (*?Contract*) in the **Individual Control Options** list.
10. Press the **Properties** button, then select the **Events** tab.
11. Highlight the *Accepted* event, then press the **Properties** button.
12. Check the **Override default action** box, then select *Partial page refresh* from the drop-down list.
13. Press the **OK** buttons on all the windows until you return to the application tree (4 times).

Make and Deploy

1. Choose Project ▸ Make (or press the Make icon button on the toolbar).

   Your Web application is ready to deploy once again.

2. Press the **OK** button on the compiler window.

3. Open Windows Explorer (or Windows NT Explorer).

4. Copy *Webtree.exe* from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

   This time you need only deploy the application, the icons have not changed.

Let's run the application to see how the changes we made affect its behavior.

See the difference

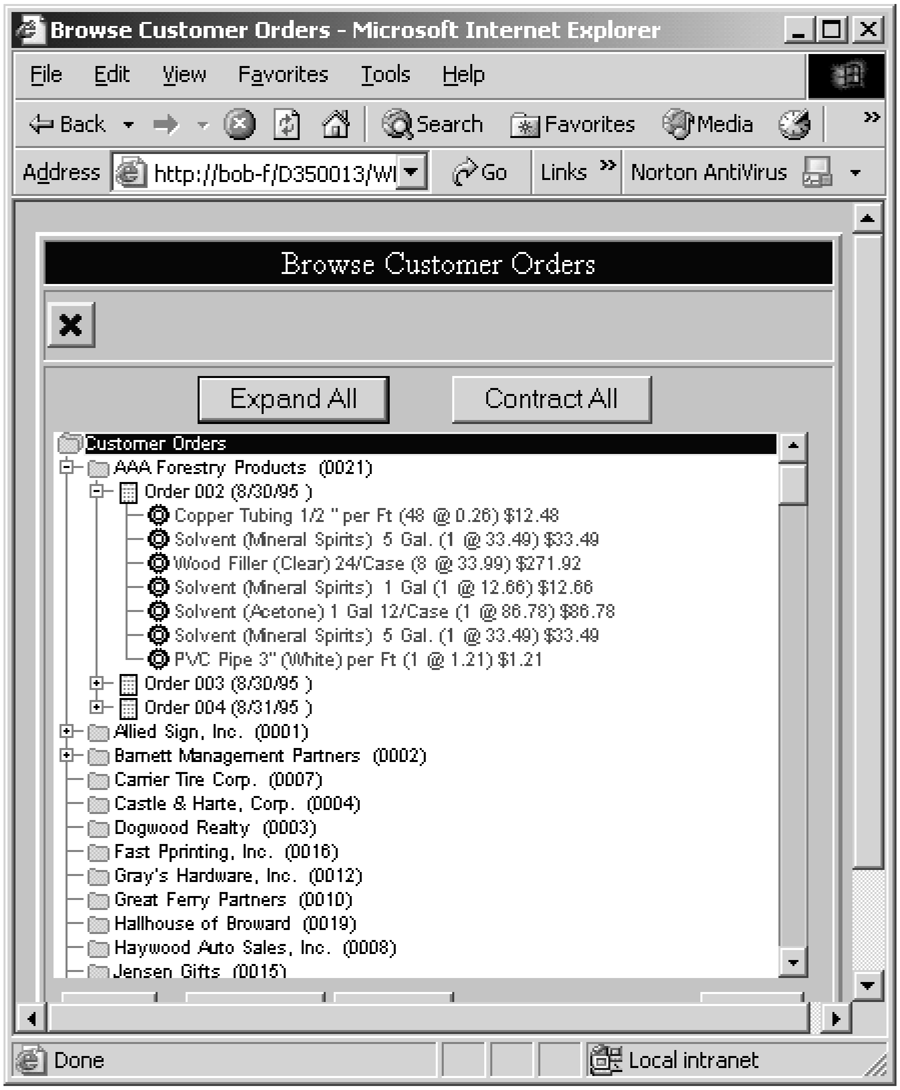
1. Task-switch back to your browser.

2. Start the application in the browser by clicking on the Restart hyperlink.

3. CLICK on the **Orders** button again.

4. CLICK on the **Expand All** and **Contract All** buttons and notice the behavior now.

   You should notice that the tree now re-displays the Tree data without sending an entire page.


Browse Customer Orders - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://bob-f/D350013/WI
Go
Links
Norton AntiVirus
Browse Customer Orders
Expand All
Contract All
Done
Local intranet

- Customer Orders
  - AAA Forestry Products (0021)
    - Order 002 (8/30/95)
      - Copper Tubing 1/2 " per Ft (48 @ 0.26) $12.48
      - Solvent (Mineral Spirits) 5 Gal. (1 @ 33.49) $33.49
      - Wood Filler (Clear) 24/Case (8 @ 33.99) $271.92
      - Solvent (Mineral Spirits) 1 Gal (1 @ 12.66) $12.66
      - Solvent (Acetone) 1 Gal 12/Case (1 @ 86.78) $86.78
      - Solvent (Mineral Spirits) 5 Gal. (1 @ 33.49) $33.49
      - PVC Pipe 3" (White) per Ft (1 @ 1.21) $1.21
    - Order 003 (8/30/95)
    - Order 004 (8/31/95)
  - Allied Sign, Inc. (0001)
  - Barnett Management Partners (0002)
  - Carrier Tire Corp. (0007)
  - Castle & Harte, Corp. (0004)
  - Dogwood Realty (0003)
  - Fast Pprinting, Inc. (0016)
  - Gray's Hardware, Inc. (0012)
  - Great Ferry Partners (0010)
  - Hallhouse of Broward (0019)
  - Haywood Auto Sales, Inc. (0008)
  - Jensen Gifts (0015)

5. Exit the application.

Congratulations! You are well on your way to developing Web applications. In the next chapter, we will discuss some advanced options you have at your disposal with Internet Connect.

9 - Tutorial— Advanced Web Programming Techniques

Now that you have learned how to create a Web application and how to port an existing Clarion application to the Web, you have all the skills you need to publish database applications on the Internet.

But, there is more you can do with Internet Connect. This chapter will show you some of the advanced techniques you can use in your Web Applications.

For the rest of the tutorial, we will continue to use the WebTree example that you used in the previous chapter.

In this chapter, you will:

- Add a Login window and use Cookies to “remember” a user’s login name the next time the app is started.
- Use a Code Template to Embed Static HTML.
- Use a Code Template to Embed Dynamic HTML using a variable.
- Use an Internet Embed point to write conditional HTML Code.
- Password protect a procedure.
- Add a Web Splash window to inform first time users that the Java Support Library is downloading.
- Use Embedded HTML to align an Image on the Web.
- Use Individual Control Options to ensure embedded source code is executed over the web.
- Use embedded source code to restrict Edit-In-Place when running over the web.

This should all take about thirty minutes. By the end of this chapter, you’ll learn most of the methods available to customize of your Web applications.

Let’s continue!

Using Cookies

In this section, we will add a login window to allow users to identify themselves. The application will use cookies to store that name and “remember” the login name. The next time the user starts the application, the prompt will not appear.

Starting Point:
You should have the WebTree.app open in the Clarion development environment.

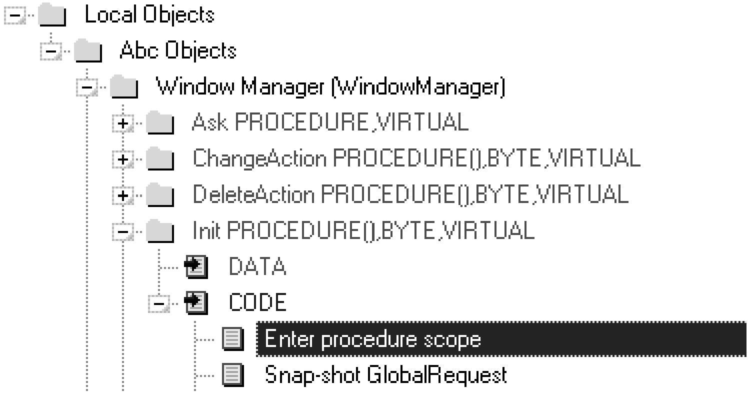
This tutorial assumes that you installed Clarion in C:\Clarion6 and the Application broker in C:\CWICWEB. If you used a different directory, you will have to modify the instructions accordingly.

Add a login procedure

1. In the Application Tree, highlight the *Main* procedure, then press the Properties icon button. This opens the Procedure Properties window.

2. Press the **Embeds** button. This opens the *Embedded Source* window.

3. Highlight the embed point as shown below:



Local Objects
Abc Objects
Window Manager (WindowManager)
Ask PROCEDURE,VIRTUAL
ChangeAction PROCEDURE(),BYTE,VIRTUAL
DeleteAction PROCEDURE(),BYTE,VIRTUAL
Init PROCEDURE(),BYTE,VIRTUAL
DATA
CODE
Enter procedure scope
Snap-shot GlobalRequest

This point ensures that the LoginWindow is called before the window opens.

4. Press the **Insert** button. This opens the *Select Embed Type* window.

5. Highlight *Call a Procedure*, then press the **Select** button.

6. In the **Procedure to call** field, type *LoginWindow*, then press the **OK** button.

7. Press the **Close** button on the *Embedded Source* window and the **OK** button on the *Procedure Properties* window.

This adds the *LoginWindow* procedure as a *ToDo* item.

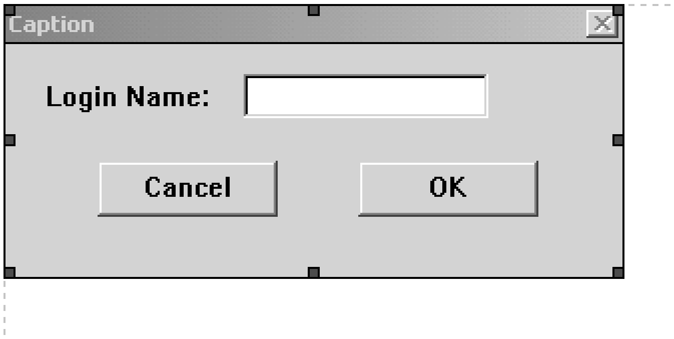
Add the login window

1. In the Application Tree, highlight the *LoginWindow* procedure, then press the **Properties** button. This opens the *Select Procedure Type* window.
2. On the **Templates** tab, highlight the *Window-Generic Window Handler*, then press the **Select** button. This opens the *Procedure Properties* window.
3. Press the **Window** button. This opens the *New Structure* window.
4. Highlight *Simple Window*, then press the **OK** button. This opens the Window Formatter.

Design the login window

1. Select **Populate** ▸ **Column**. This opens the *Select Column* dialog.
2. In the **Tables** list on the left, highlight *Global Data*, then in the **Columns** list on the right, select *LoginName*, then press the **Select** button.

   This variable was created for you in the example application.
3. CLICK on the window to populate the Prompt and Entry control.
4. Select **Populate** ▸ **Control Template**. This opens the *Select Control Template* window.
5. Highlight *CancelButton* then press the **Select** button.
6. CLICK on the window to populate the Cancel button control.
7. Select **Populate** ▸ **Control Template**. This opens the *Select Control Template* window.
8. Highlight *CloseButton* then press the **Select** button.
9. CLICK on the window to populate the Close button control.
10. Change the text of the the Close button control to *OK*.



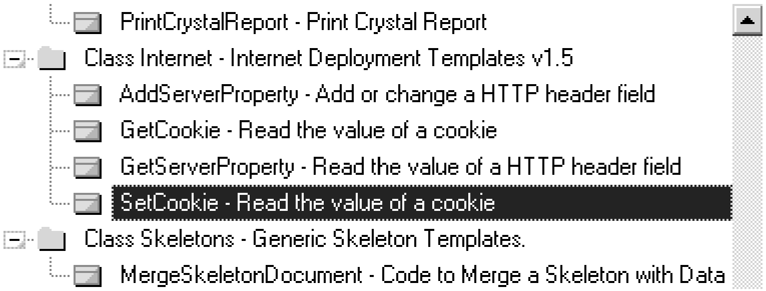
11. Reposition the controls on the window as you see fit.

Add the “Cookie” code to save the LoginName

1. DOUBLE-CLICK on the **OK** button control to access the Embedded Source points for the control.

2. Highlight the *Control Events, ?Close, Accepted, Genertated Code* embed point then press the *Insert* button. This inserts the code after any generated code for the control.

3. Select the *SetCookie* code template then press the **Select** button.



- PrintCrystalReport - Print Crystal Report
- Class Internet - Internet Deployment Templates v1.5
  - AddServerProperty - Add or change a HTTP header field
  - GetCookie - Read the value of a cookie
  - GetServerProperty - Read the value of a HTTP header field
  - SetCookie - Read the value of a cookie
- Class Skeletons - Generic Skeleton Templates.
  - MergeSkeletonDocument - Code to Merge a Skeleton with Data

4. In the **Cookie name** field, type *LoginName*.

5. In the **New Value** field, type *LoginName* (or select the LoginName global variable from the File schematic using the ellipsis button).

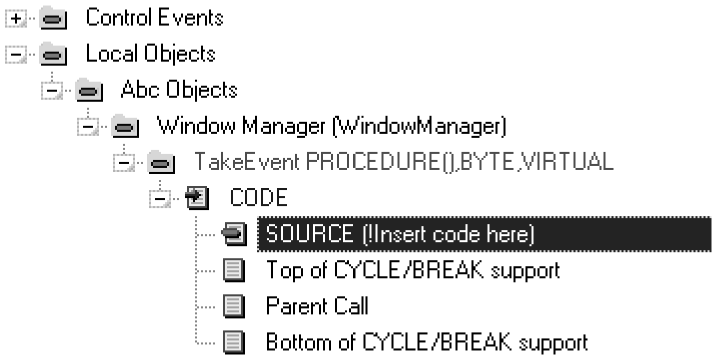
6. Press the **OK** button.

7. Press the **Close** button on the *Embedded Source* window.

Add the “Cookie” code to get the LoginName

1. DOUBLE-CLICK on the window to access the *Embedded Source* points for the window.

2. Highlight the embed point as shown below then press the **Insert** button.



- Control Events
- Local Objects
  - Abc Objects
    - Window Manager (WindowManager)
      - TakeEvent PROCEDURE(),BYTE,VIRTUAL
        - CODE
          - SOURCE (!Insert code here)
          - Top of CYCLE/BREAK support
          - Parent Call
          - Bottom of CYCLE/BREAK support

3. Highlight *Source* then press the **Select** button.

4. Type in the source code below:

```
CASE EVENT()
OF Event:NewPage                                 !New Page is requested
  LoginName = Broker.Http.GetCookie('LoginName') !get login cookie
  DISPLAY                                        !refresh
OF EVENT:CloseWindow
  RETURN PARENT.TakeEvent()                      !process the close event
END
IF LoginName                                     !If cookie exists
  POST(Event:CloseWindow)                        !close this page
END
```

This code “gets” a cookie when the window is active. If it sucessfully retrieves a cookie and sets the LoginName variable, it closes the window (before the user sees it). This means a user only needs to login once, then the server “recognizes” the user the next time around.

5. Exit the Source editor and save the changes.

6. Press the **Close** button on the *Embedded Source* window.

7. Exit the Window Formatter and save the changes.

8. Press the **OK** button on the *Procedure Properties* window.

Make and Deploy

1. Choose **Project** ▸ **Make** (or press the Make icon button on the toolbar). Your Web application is ready to deploy.

2. Press the **OK** button on the compiler window.

3. Open Windows Explorer (or Windows NT Explorer).

4. Copy *WebTree.exe* from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

Run the application

1. Start the Application Broker by DOUBLE-CLICKing on *C60APS10.exe* (or *C60APS.exe* if you have the full version of the Application Broker) in the *C:\CWICWEB\* directory.

2. Start your browser.

3. Start the WebTree application in the browser (http://localhost/exec/webtutor/webtree.exe.0).

Examine the application

The first time you run the application. You are prompted to provide a login name. The next time you run it, you are not prompted, because the system reads your cookie and the value of the global variable is set to the value in the cookie.

1. Type in a name when the Login screen appears then press **OK** .
2...Exit the application

3. Restart the WebTree application in the browser. Notice that the second time, you are not prompted to log in.

4. Exit the application

Leave your browser open with the restart page displayed. You will use this to restart your application after making some changes.

Let's make another change to the application to display the user's LoginName using the *Dynamic HTML* code template.

Embedding HTML

One of the most powerful features of the Internet Developer's Kit is the ability to embed HTML code in the HTML pages which are output by the Web-enabled application.

When you embed HTML code (using the special embed points added by the Global Internet Application Extension template), it is inserted at the specified location in the HTML returned to the browser which executed the application.

Starting Point:
You should have the Clarion development environment open and open the WebTree.app application.

Adding Dynamic HTML using a variable

We have written the code needed to set and retrieve a user's login name and store it in a global variable. Now we will display that name on the Web page below the HTML representation of the window.

1. In the Application Tree, highlight the *Main* procedure, then press the **Properties** icon button. This opens the *Procedure Properties* window.
2. Press the **Embeds** button.

   This opens the *Embedded Source* window.
3. Highlight the *Internet-Before the Closing </BODY> tag* embed point, then press the **Insert** button. This opens the *Select Embed Type* window.
4. In the code template section, highlight *Dynamic HTML*, then press the **Select** button.
5. In the **Dynamic Text** field, type the following:

   ```
   '<<P>' & CLIP(LoginName) & ' is logged in <</P>'
   ```
6. Press the **OK** button on the code template window.
7. Press the **Close** button on the *Embedded Source* window and the **OK** button on the *Procedure Properties* window.

Make and Deploy

1. Choose **Project** ▸ **Make** (or press the **Make** icon button on the toolbar).

   Your Web application is ready to deploy.

2. Press the OK button on the compiler window.
3. Open Windows Explorer (or Windows NT Explorer).
4. Copy *WebTree.exe* from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

Examine the application

1. Restart the WebTree application in the browser (click on the Restart hyperlink).

   If you have already run the application on this machine, you will not be prompted to Log In. Instead, the server reads your “cookie” and sets the LoginName global variable to that value. The LoginName variable now displays on the Web page below the toolbar buttons.

2. Exit the application.

   Leave your browser open with the restart page displayed. You will use this to restart your application after making some changes.

Let’s make some more changes to the application using Embedded HTML.

Adding Static HTML

In the last section, we added HTML code that was constructed using a combination of text and variables. In this section we will use the Static HTML code template to add HTML code that will remain static.

We will use this to add a link at the bottom of the page that will allow users to Email the Webmaster with comments or questions about the application.

1. In the Application Tree, highlight the *Main* procedure, then press the **Properties** button. This opens the *Procedure Properties* window.
2. Press the **Embeds** button.This opens the *Embedded Source* window.
3. Highlight the *Internet-Before the Closing </BODY>* tag embed point and press the **Insert** button.

   This opens the *Select Embed Type* window.

4. Highlight *Static HTML*, then press the **Select** button.

5. In the **HTML to Insert** box, type the following:

   ```
   <P><A HREF="mailto:nobody@softvelocity.com">Comments?</A></P>
   ```

6. Press the **OK** button on the code template window.

7. Press the **Close** button on the *Embedded Source* window and the **OK** button on the *Procedure Properties* window.

Make and Deploy

1. Choose **Project** ▸ **Make** (or press the Make icon button on the toolbar).

   Your Web application is ready to deploy.

2. Press the **OK** button on the compiler window.

3. Open Windows Explorer (or Windows NT Explorer).

4. Copy *WebTree.exe* from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

Examine the application

1. Restart the WebTree application in the browser (click on the Restart hyperlink).

   You will notice the new link. If you click on the link, your browser opens your Email client with a new preaddressed message.

2. Exit the application.

   Leave your browser open with the restart page displayed. You will use this to restart your application after making some changes.

Adding conditional HTML in Clarion Source Code

A third method of inserting embedded HTML into your Web pages is by using the `Target.WriteLn` method in embedded source code. This allows you to use Clarion code to write the HTML code. One benefit of using Clarion code is the ability to control the HTML code you want to write. In other words, you can utilize the logical structures in the Clarion language to control what is written. You can write one line or another using an IF..THEN..ELSE clause, or a CASE structure.

We will use this technique to display a random advertisement on the bottom of the page using an EXECUTE structure.

1. In the Application Tree, highlight the *Main* procedure, then press the **Properties** icon button. This opens the *Procedure Properties* window.

2. Press the **Embeds** button. This opens the *Embedded Source* window.

3. Highlight the *Internet-Before the Closing </BODY> tag* embed point then press the **Insert** button. This opens the *Select Embed Type* window.

4. Highlight *Source*, then press the **Select** button.

5. In the *Embedded Source* editor, type the following source code:

```
Str1" = '<<A HREF="http://www.'
Str2" = '.com"><<IMG SRC="'
Str3" = '" BORDER=0><</A>'

EXECUTE RANDOM(1,5)
Target.WriteLn(CLIP(Str1") &'softvelocity'& CLIP(Str2") & SELF.Files.GetAlias('1.GIF') & Str3")
Target.WriteLn(CLIP(Str1") & 'icetips'    & CLIP(Str2") & SELF.Files.GetAlias('2.GIF') & Str3")
Target.WriteLn(CLIP(Str1") & 'finatics' & CLIP(Str2") & SELF.Files.GetAlias('3.GIF') & Str3")
Target.WriteLn(CLIP(Str1") & 'flpanthers' & CLIP(Str2") & SELF.Files.GetAlias('4.GIF') & Str3")
Target.WriteLn(CLIP(Str1") & 'flamarlins' & CLIP(Str2") & SELF.Files.GetAlias('5.GIF') & Str3")
END
```

Note:

You can copy and paste this text from *chap4.txt* in the \webtutor directory.

6. Exit the Source editor and save the changes.

7. Press the **Close** button on the *Embedded Source* window and the **OK** button on the *Procedure Properties* window.

Make and Deploy

1. Choose **Project** ▸ **Make** (or press the **Make** icon button on the toolbar).

   Your Web application is ready to deploy.

2. Press the **OK** button on the compiler window.

3. Open Windows Explorer (or Windows NT Explorer).

4. Copy *WebTree.exe* from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

5. Copy the GIF files (*.gif) from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\Public* directory.

Examine the application

1. Restart the WebTree application in the browser (click on the Restart hyperlink).

   You will notice the new image and link. Each time you start the application, a random ad appears.

2. Exit the application.

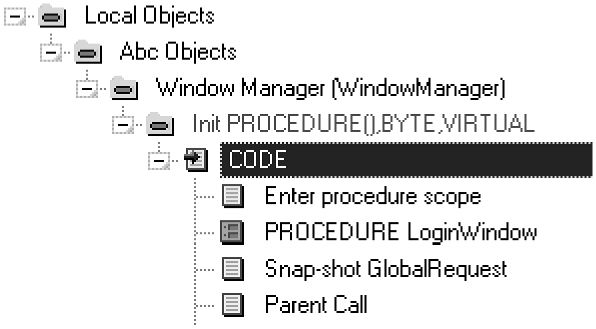
   Leave your browser open with the restart page displayed. You will use this to restart your application after making some changes.

Covering the Download with a Splash Window

In order for a browser to “run” a Web-enabled application, the Java Support Library (JSL) must be available to the client browser. First-time users must download either Clarion.CAB (for Microsoft Internet Explorer) or Clarion.ZIP (for Netscape). In most browsers, the JSL is only downloaded once and remains cached (until the user clears that cache). Although the JSL is very compact for the degree of functionality it provides, it can still take several minutes to download over a 28.8 modem. With that in mind, we will use a “splash screen” window to alert first-time users that the download is in progress. By placing a Java Button on that window, we can prevent users from continuing until the JSL is downloaded and the Java button is initialized.

Starting Point:
You should have the Clarion development environment open and open the WebTree.app application.

1. In the Application Tree, highlight the *Main* procedure, then press the **Properties** icon button. This opens the *Procedure Properties* window.
2. Press the **Embeds** button. This opens the *Embedded Source* window.
3. Highlight the embed point as shown below:



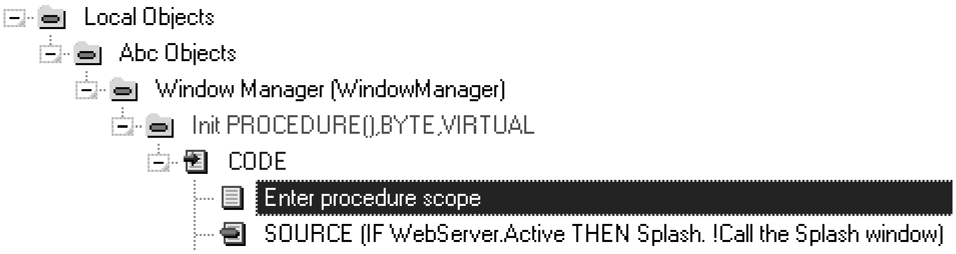
Local Objects
  Abc Objects
    Window Manager (WindowManager)
      Init PROCEDURE(),BYTE,VIRTUAL
        CODE
          Enter procedure scope
          PROCEDURE LoginWindow
          Snap-shot GlobalRequest
          Parent Call

4. Press the **Insert** button. This opens the *Select Embed Type* window.
5. Highlight *Source*, then press the **Select** button.
6. In the Embedded Source editor, type the following source code:

```
IF WebServer.Active THEN Splash.
```

This makes sure that the Splash procedure is only called when the application is running over the Web.

7. Make sure this embed is listed before the call to the LoginWindow procedure using the up or down button.



Local Objects
Abc Objects
Window Manager (WindowManager)
Init PROCEDURE(),BYTE,VIRTUAL
CODE
Enter procedure scope
SOURCE (IF WebServer.Active THEN Splash. !Call the Splash window)

This ensures that the *Splash* procedure is called before any other window opens.

8. Press the **Close** button on the *Embedded Source* window.

9. Press the **Procedures** button. This opens the *Called Procedures* window.

10. Highlight *Splash*, then press the **OK** button.

This connects the *Splash* procedure to the Main procedure in the Application Tree. This is necessary if your application is using Local MAPs.

Changing the BUTTON to a Java Button

The Splash window contains some text, a button, and an IMAGE control. The BUTTON was populated as a CloseButton control template with the text changed to Continue. Since the button is created as an HTML button by default, you will specify otherwise. We want it to be a Java button so that it will not be available to the end user until the JSL has downloaded.

1. In the Application Tree, highlight the *Splash* procedure, then press the **Properties** icon button.

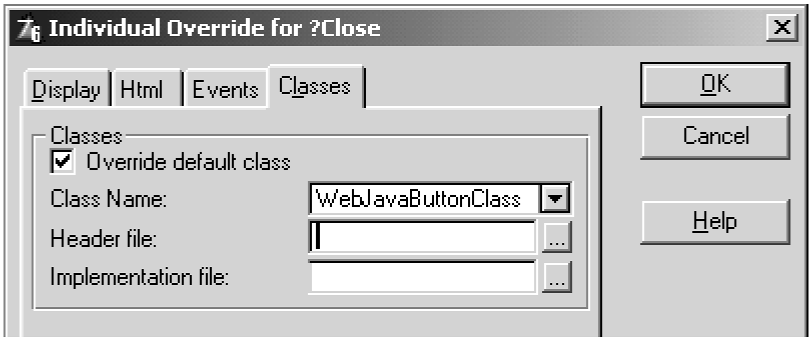
2. Press the **Internet Options** button.

3. Select the **Controls** tab.

4. Highlight *?Close* in the **Individual Control Options** list, then press the **Properties** button.

5. Select the **Classes** tab.

6. Check the **Override default Class** box, then select the *WebJavaButtonClass* from the **Class Name** drop-down list.



Classes

☑ Override default class

Class Name: WebJavaButtonClass

Header file:

Implementation file:

7. Press the **OK** button. Leave the *Internet Options* window open. We will use it in the next section.

Centering the Image on the Splash window

The *Splash* window's IMAGE control is positioned so that is is centered horizontally in the window. This portion of the tutorial will add some HTML code to ensure that the IMAGE remains centered when running over the Web.

1. Highlight *?Image1* in the **Individual Control Overrides** list, then press the **Properties** button.

2. Select the **HTML** tab.

   This window allows you to enter HTML code before and after a control. This HTML code only affects the control when running over the Web.

3. In the **HTML before control** box, type:
.... <CENTER>

4. In the **HTML after control** box, type:

   `</CENTER>`

5. Press the **OK** buttons on all the windows until you return to the Application Tree (3 times).

Make and Deploy

1. Choose **Project** ▸ **Make** (or press the **Make** icon button on the toolbar).

   Your Web application is ready to deploy.

2. Press the **OK** button on the compiler window.

3. Open Windows Explorer (or Windows NT Explorer).

4. Copy *WebTree.exe* from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

Examine the application

1. Restart the WebTree application in the browser (click on the Restart hyperlink).

   You will notice the Splash window now appears before any other window.

2. Exit the application.

   Leave your browser open with the restart page displayed. You will use this to restart your application after making some changes.

Using Partial Refresh to Update Controls

In Windows applications, programmers often embed code to update one control when the value of another control changes. For example, you might embed code to change the total of a line item when the quantity of items changes. The Webtree application has code like this in the *UpdateItems* procedure. The embedded code is tied to the EVENT:Accepted on each control. In other words, when the user changes the value in a control and tabs off it or selects another control with a mouse click, the code is executed.

When an application runs over the Web, ENTRY controls are processed on the browser by default. In other words, there is no interaction between the browser and the server application—unless you change the event handling options for that control. In this section, you will change the action for three controls to ensure that embedded code is executed on the server for an Event:Accepted for these controls.

1. In the Application Tree, highlight the *UpdateItems* procedure, then press the **Properties** icon button. This opens the *Procedure Properties* window.
2. Press the **Internet Options** button.
3. Select the **Controls** tab.
4. Highlight *?ITEM:ProdCode* in the **Individual Control Options** list, then press the **Properties** button.
5. Select the **Events** tab.
6. Highlight *Accepted*, then press the **Properties** button.
7. Check the **Override default action** box, then select the *Partial page refresh* from the **Action on Event** drop-down list.
8. Press the **OK** buttons on all the windows until you return to the *Internet Options* window (twice).
9. Repeat the last 5 steps for *?ITEM:Quantity* and *?ITEM:Price*.
10. Press the **OK** buttons on all the windows until you return to the Application Tree (twice).

Make and Deploy

1. Choose **Project** ▸ **Make** (or press the **Make** icon button on the toolbar).

   Your Web application is ready to deploy.

2. Press the **OK** button on the compile results window.
3. Open Windows Explorer (or Windows NT Explorer).
4. Copy *WebTree.exe* from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

Examine the application

1. Restart the WebTree application in the browser (click on the Restart hyperlink).
2. Press the **Orders** button.
3. Press the **Expand All** button.
4. Highlight one of the line itens (the green lines).
5. Press the **Change** button
6. Change the amount in the Quantity Field, then press TAB.

   Notice the Extended Total changes. If you change the Price field or Product Code, the Extended Total also changes.

7. Exit the application.

Leave your browser open with the restart page displayed. You will use this to restart your application after making some changes.

Restricting Access to a Procedure

For the next part of the tutorial we will restrict access to a procedure using the browser’s built-in authentication support and the Internet Procedure Extension template’s password protection capabilities. When a password protected procedure is called, the browser's authentication window displays. You do not need to create a window to collect login information. Password protection is based on an area, a username and a password. The “area” is the protected procedure.

The browser prompts the user for a user name, and a password. These are then sent to the program for validation. If the program accepts the password (i.e., it RETURNs TRUE from the `WebWindow.ValidatePassword` method), the new page is displayed, otherwise the browser prompts again. After three attempts the browser displays a message informing the user that access is denied. This page automatically returns the user to the last active place in the program after a few seconds.

Note:

If the page has already been visited in the current session the browser will normally supply the user name and the password without prompting the user. This feature is built-in to most browsers.

There are a few methods of password protection (see *Using Passwords* in the *Web Application Design Considerations* chapter). We will use the more advanced method—to override the `WebWindow.ValidatePassword` method.

Starting Point:
You should have the Clarion development environment open and open the WebTree.app application.

Password Protection

To implement password protection that is validated against a data file, you must add the validation file to the file schematic, add the password challenge in the Procedure Extension template, and override the `WebWindow.ValidatePassword` method with your validation code.

Add the Validation File

1. In the Application Tree, highlight the *UpdateProduct* procedure, then press the **Properties** icon button. This opens the *Procedure Properties* window.
2. Press the **Tables** button. This opens the *Table Schematic Definition* window.
3. Highlight the *Other Files*, then press the **Insert** button. This opens the Select File window.

4. Highlight *Userlist*, then press the **Select** button.

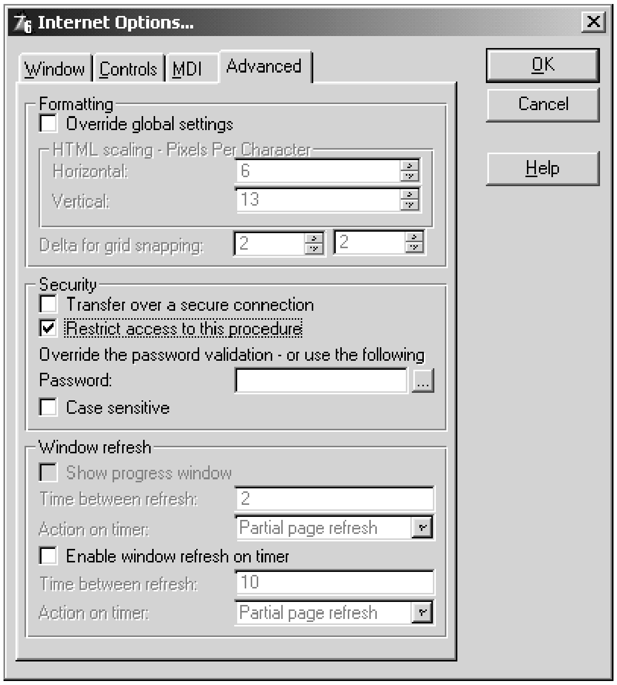
5. Press the **OK** button on the *Table Schematic Definition* window.

Add the Password Challenge

1. Press the **Internet Options** button.

2. Select the **Advanced** Tab.

3. Check the **Restrict access to this procedure** box.



Formatting
☐ Override global settings
HTML scaling - Pixels Per Character
Horizontal: 6
Vertical: 13
Delta for grid snapping: 2 2

Security
☐ Transfer over a secure connection
☑ Restrict access to this procedure
Override the password validation - or use the following
Password:
☐ Case sensitive

Window refresh
☐ Show progress window
Time between refresh: 2
Action on timer: Partial page refresh
☐ Enable window refresh on timer
Time between refresh: 10
Action on timer: Partial page refresh

4. Press the **OK** button.

5. Press the **Embeds** button. This opens the *Embedded Source* window.

6. Highlight the *Internet- Password Validation Code Section* embed point then press the **Insert** button. This opens the *Select Embed Type* window.

   By entering code into the *Internet- Password Validation Code Section* embed point you are overriding the default method for password validation.

This embed point generates inside a method with two parameters: UserName and Password, which it receives from the browser. The method should return TRUE if the password is valid, and FALSE if it is not valid. This allows you to look up the information in a file, or use any other method you choose to validate the password.

7. Highlight *Source*, then press the **Select** button.

8. In the *Embedded Source* editor type the following source code:

```
USE:UserID = UserName
IF Access:UserList.Fetch(USE:KeyUserID) !lookup UserName in file
 RETURN(False)
END
IF USE:UserPassword = Password          !Check the password
 RETURN(True)
ELSE
 RETURN(False)
END
```

9. Exit the Source editor and save the changes.

10. Press the **Close** button on the *Embedded Source* window and the **OK** button on the *Procedure Properties* window.

Make and Deploy

1. Choose **Project** ▸ **Make** (or press the **Make** icon button on the toolbar).

   Your Web application is ready to deploy.

2. Press the **OK** button on the compile results window.

3. Open Windows Explorer (or Windows NT Explorer).

4. Copy *WebTree.exe* from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

Examine the application

1. Restart the WebTree application in the browser (CLICK on the Restart link).

2. Press the **Products** button.

3. Press the **Insert** button to add a new product.

   The Browser's authentication window appears.

4. In the UserName field, type *Fred*.

5. In the Password field, type *Wilma*.

   The values you entered are in the Userlist file. This file was precreated with two users. Note that there is no procedure in this application to edit this file. This is a common method of handling user password files where only a system administrator has permission to add users. Feel free to create procedures to update this file as you see fit.

6. Exit the application.

Restricting Edit-In-Place

The ABC Templates in Clarion allow you to enable Edit-In-Place with a single checkbox. This feature, however, is not supported when running over the Web. Over the Web, you must have a separate Form for updates. There is a simple method to alternate between edit-in-place when running locally in Windows and calling a form when running over the Web.

If you enable Edit-In-Place *and* specify an update procedure with the BrowseBox control template, you have two-thirds of your work done. The template generated code either calls a separate update procedure or does edit-in-place depending on the value of the **`BRWn.AskProcedure`** property. Set the **`BRWn.AskProcedure`** property to 0 (zero) and you have Edit-in-Place; Set it to 1 (One) and you call the update procedure.

To use Edit-in-place for local Windows and a form when running over the Web:

1. Select the *BrowseProduct* procedure, then press the **Properties** icon button.

2. In the UpdateButton section of the *Procedure Properties* window, check the **Use Edit in Place** box.

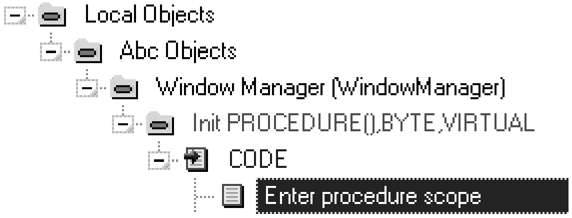
   Notice that an update procedure is already specified. Make sure to leave that procedure named.

   Next, embed the code to set the update action to call Edit-in-Place when running in Windows and call the form when running over the Web.

3. Press the **Embeds** button.

   This opens the Embedded Source window.

4. Highlight the embed point as shown below then press the **Insert** button.

5. Highlight *Source*, then press the **Select** button.

6. In the Embedded Source editor, type the following source code:

```
IF WebServer.Active
 BRW1:AskProcedure = 1
END
```

7. Exit the Source editor and save the changes.
8. Press the **Close** button on the *Embedded Source* window and the **OK** button on the **Procedure Properties** window.

Make and Deploy

1. Choose **Project** ▸ **Make** (or press the **Make** icon button on the toolbar).

   Your Web application is ready to deploy.
2. Press the **OK** button on the compile results window.
3. Open Windows Explorer (or Windows NT Explorer).
4. Copy *WebTree.exe* from the *C:\Clarion6\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

Examine the application

1. Restart the WebTree application in the browser (click on the Restart hyperlink).
2. Press the **Products** button.
3. Press the **Insert** button to add a new product. The Browser's authentication window appears.
4. In the UserName field, type *Fred*. In the Password field, type *Wilma*. Notice that the **Update Products** form appears.
5. Exit the application.
6. Run the application under Windows.
7. Press the **Products** button.
8. Press the **Insert** button to add a new product. Notice that Edit-In-Place is now active.
9. Exit the application.

Congratulations! You have sucessfully completed the tutorial portion of this manual. You should have enough experience now to create robust Web database applications.

The rest of the book explains the IBC Templates, the IBC Library, and application design tips and techniques. Read on.

10 - The Internet Builder Class Templates

This chapter covers the Internet Builder Class (IBC) Templates in the Internet Developer’s Kit. These templates are designed to work with both of the template chains included in Clarion (ABC and Clarion). For the most part, the IBC Templates work in the same manner when used with either template chain. The differences are noted in the section where those differences appear.

The IBC Templates are made up of a single Global Application extension template, a procedure template, and several code templates.

The Global Internet Application Extension template automatically adds the Procedure extension template to every procedure in the application. This allows you to Web-enable an entire application in a single step.

The combination of global and procedure level settings provides customization capabilities at either level. To make a setting application-wide, you set a Global option. To specify an option for a single procedure, you make the setting for that procedure. Many of the Global and Procedure settings are the same; the only difference is the *scope* of the setting.

The Global Internet Application Extension Template

The Global Internet Application Extension Web-enables a Clarion application. It adds the functionality of generating dynamic HTML when the application is accessed through the Application Broker. This template allows you to specify the options to use when generating an HTML representation of your windows and reports.

In addition, it automatically adds the Internet Procedure Extension to every procedure in your application and any procedures subsequently added to the application. The Procedure extension allows you to override many of the global options for a specific procedure.

This template allows you to customize the appearance and behavior of your application when it is executed over the Web. The settings you specify here are global in nature; that is, they affect every procedure in your application.

You can override most of these settings on a procedure level using the Internet Procedure Extension’s settings. In addition, some options can be specified on a control-by-control basis. The combination of these three levels of customization provides you with complete flexibility of design.

Note:

None of these settings affect your application when running locally as a Windows executable.

Page Settings

When run over the Web, an application’s current window is displayed inside an HTML page (a Web page). The page settings allow you to specify a background color or background image for the HTML page. The template generated code calls the `WebWindow.SetPageBackground` method to set these properties.

Center Window on Page

Check this box to center the HTML representation of your window inside the Web page. This adds `<CENTER></CENTER>` HTML tags to the Web page.

Background color

You can specify the color to use for the Web page. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button. The default is no color (the equate is COLOR:NONE). This means that the browser's default page color is used.

Background image

You can specify an image to display as the background for the Web page. Specify an image filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is no image.

Window Settings

When run over the Web, an application’s current window is represented by an HTML <TABLE>. This allows you to set <TABLE> properties such as background color and border width. The prompts on this tab allow you to specify the appearance of the “window” (<TABLE>) portion of the HTML page. The template generated code calls the `WebWindow.SetBackground` method to set these properties.

Background color

You can specify the color to use for your application’s window. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button. The default is no color (the equate is COLOR:NONE). This means that the background color of the application’s window is used.

Tip

You can also set colors for discrete parts of the window, such as the toolbar. See *Window Component Options*.

Background image

You can specify an image to display as the background for your application’s window. Specify an image filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is no image.

Tip

A background image tiles (i.e., it repeats as many times as its size allows) inside an HTML <TABLE> cell representing the application’s window. Provide a small image that tiles to save bandwidth.

Window border width

Specify the border width for your application’s window. The default is 2, which makes a thin border. Specify a 0 border width to display no border. The template generated code calls the WebWindow.SetBorderWidth method to set the property.

Help

Enable Help for internet applications

Check this box to enable links from Help buttons in your application. (A Help button is a BUTTON with the STD:Hlp attribute). If Help is enabled, a Help button will call a Web page based on the Help ID of the current window. This document is opened in a Browser window named “_HELP” which will cause a new browser window to open or if a frame already has that name, it displays the Help document inside that frame. The template generated code uses the `WebWindow.SetHelpDocument` method or the `WebWindow.SetHelpURL` method to set the properties you specify. You are responsible for creating the corresponding HTML pages. See *Implementing Help in your Web Application*.

URL of Help documents

The base location of the HTML files for your Help. For example, your HTML Help files are located in a separate subdirectory.

Help Window Style

You can optionally supply a style for your Help window.

Help Ids are links within a base document

If your Help is designed as a single document with mid-page anchors, check this box. If not checked, the Help buttons reference individual HTML pages.

Help Document

The base document containing the mid-page anchors. This field is enabled only when the Help Ids are links within a base document box is checked.

Window Components

Press this button to specify the appearance of the window’s components (e.g., TOOLBAR, MENU, and Caption areas). See *Window Component Options*.

Control

The prompts on this tab allow you to set the defaults for generating the HTML code that represents each of your application’s controls.

Tip

In addition to the settings here, you can set control options for individual controls in the procedure template’s Internet Options. See Individual Overrides for a Control.

General

If control disabled
Specifies what to display on the browser when a window control is disabled. This option is provided because most HTML controls do not support disabling. This sets the WebWindow.DisabledAction property.The choices are:

> *Hide*
> Hides any disabled controls (the default).
>
> *Hide if cannot disable*
> Hides any disabled control when it cannot be disabled on the Web page. Most HTML controls cannot be disabled.
>
> *Show*
> Displays any disabled controls. It appears normally (i.e., it will appear to be enabled), but changes made to the control will not affect the underlying application.

Drop listboxes - Replace with Java non-drop list
Allows you to replace a drop-down list with a page-loaded Java Listbox. If your drop-down lists need to display more than one column, use this option.

Sheets - Border width
Specify the border width for SHEET controls. The default is 2, which makes a thin border. Specify a 0 border width to display no border. This sets the **`WebWindow.SheetBorderWidth`** property.

Options - Border width
Specify the border width for OPTION controls. This only applies to OPTIONs with the BOXED attribute. The default is 2, which makes a thin border. Specify a 0 border width to display no border. This sets the **`WebWindow.OptionBorderWidth`** property.

Groups - Border width
Specify the border width for GROUP controls. This only applies to GROUPs with the BOXED attribute. The default is 2, which makes a thin border. Specify a 0 border width to display no border. This sets the **`WebWindow.GroupBorderWidth`** property.

MDI

This section determines the manner in which Application Menus and Toolbars are handled.

Tip

For control over specific Menu or Toolbar items, set the MDI overrides in the Frame Procedure's Internet Options.

Frame Menu

This section determines the manner in which Application Menus are handled. This allows you to specify which global menu options are displayed on "child" windows.

Include on Child Windows
Select an option from the drop-down list. The choices are:

All Menu Items	All menu choices appear on child windows.
No Menu Items	No menu choices appear on child windows.

Ignore code in frame's ACCEPT loop
Check this box to ignore any code in the Application Frame's ACCEPT loop for menu items. If not checked, any embedded code implemented in the Frame's ACCEPT loop is automatically implemented in the child procedure.

Frame Toolbar

This section determines the manner in which Application Toolbar controls are handled. This allows you to specify which global Toolbar controls are displayed on "child" windows.

Include on Child Windows
Select an option from the drop-down list. The choices are:

All Toolbar Items All Toolbar items appear on child windows.

Standard Toolbar Only
Only the Standard Toolbar items appear on child windows. These are the buttons added by the FrameBrowseControl template.

No Toolbar Items No Toolbar items appear on child windows.

Ignore code in frame’s ACCEPT loop
Check this box to ignore any code in the Application Frame’s ACCEPT loop for toolbar items. If not checked, any embedded code implemented in the Frame’s ACCEPT loop is automatically implemented in the child procedure.

Advanced

Horizontal Pixels per Char
The number of pixels to consider for a character’s width when calculating the size to create Java applets and Images.

Vertical Pixels per Char
The number of pixels to consider for a character’s height when calculating the size to create Java applets and Images.

Note:

The numbers specified affect the overall appearance of the generated HTML page. For example, increasing the value of Vertical Pixels per Char will make the HTML Table cells taller.

Delta for grid snapping
The number of pixels to consider before repositioning a control. Specify a value for X and a value for Y. Any time a control is within this range, it is not repositioned.

Page to return to on exit
Optionally, specify the HTML page to return to when the program ends. The template generated code calls the `WebServer.Init` method to set the `WebServer.PagetoReturnTo` property.

Time out (seconds)
This specifies the maximum amount of idle time (measured in seconds) before an application closes. The default is 600 seconds (10 minutes). The template generated code calls the `WebServer.Init` method to set the `WebServer.TimeOut` property.

Sub directory for pages
The directory in which the application creates temporary directories (a temporary directory is made for each active connection) to write the dynamic HTML and graphic files. This is also the directory in which to deploy graphic files. If you provide a graphic in this directory, it is not extracted and written to the temporary directory. This defaults to /PUBLIC. The template generated code calls the `WebFilesManager.Init` method to set the property. It is not appropriate to set this property at runtime.

Classes Local to Application Broker
This specifies that the Java Support Library files are located in the /PUBLIC directory below the broker directory. If you are using multiple servers, you may want a single source from which these files are to be retrieved. In that case, you would clear the checkbox and designate the URL for the Java Support Library files. This sets the `WebServer.JavaClassPath` property.

Use Long Filenames
Check this box to allow long filnames to be created on the Web server.

Classes

The Classes Tab lets you specify which classes (objects) the Templates use to accomplish various tasks, and the source modules that contain the class definitions. This approach gives you the capability to use as much of the IBC Library as you want and as much of your own classes as you want.

To change the class for an item or override the class, highlight it in the list, then press the Properties button.

The *Internet Builder Class Library Reference* (on CD in .PDF format) is a complete guide to the classes used by the IBC templates. It provides descriptions of all the classes, methods, and properties with examples for each.

See Also: *Class Overrides*, *Global Window Component Options*

Global Window Component Options

Caption

This is the area at the top of the “window” in the HTML page. This is the portion representing the title bar.

Include caption
Check this box to display the Caption. If not checked, the caption is not used. This sets the `WebWindow.CreateCaption` property.

Background color
You can specify the color to use for the Caption area. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button. The default is Navy Blue (the equate is COLOR:Navy). If no color is specified here and you specified a Window background color in Window settings above, that color is used. If neither is specified and the application’s WINDOW has a COLOR attribute, that color is displayed in the browser. The template generated code calls the `WebCaption.SetBackground` method to set this property.

Background image
You can specify an image to display as the background for the Caption area. Specify an image filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is no image. The template generated code calls the WebCaption.SetBackground method to set this property.

Tip

A background image tiles (i.e., it repeats as many times as its size allows) inside an HTML <TABLE> cell representing the application’s window caption area. Provide a small image that tiles to save bandwidth.

Alignment
You can control the alignment of the text in the caption area. The choices are Left, Center, or Right justification. The default is Center. This sets the `WebCaption.Alignment` property.

Font family name
This allows you to specify the typeface to display. Keep in mind that the browser can only display fonts which are installed on the client’s machine. However most operating systems support font substitution and will display the closest matching font. The default is none which uses the browser’s default font. The template generated code calls the `WebCaption.SetFont` method to set this property.

Font size

Optionally, specify the point size for the Font displayed in the caption area. The default is none which uses the browser's default font size. The template generated code calls the `WebCaption.SetFont` method to set this property.

Font color

You can specify the Font's color for the Caption area. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button. The default is white (the equate is COLOR:White).

Menu

This is the menu area at the top or side of the "window" in the HTML page.

Background color

You can specify the color to use for the Menu area. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button. If no color is specified here and you specified a Window background color in Window settings above, that color is used. If neither is specified and the application's WINDOW has a COLOR attribute, that color is displayed in the browser. The template generated code calls the `WebMenubar.SetBackground` method to set this property.

Background image

You can specify an image to display as the background for the Menu area. Specify an image filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is no image. The template generated code calls the `WebMenubar.SetBackground` method to set this property.

Tip

A background image tiles (i.e., it repeats as many times as its size allows) inside an HTML <TABLE> cell representing the application's menu area. Provide a small image that tiles to save bandwidth.

Alignment

You can control the position of the menu. The choices are Above Toolbar (the default), Left of Window, or below the Toolbar. When you use Above Toolbar, the menu is spread horizontally across the top of the HTML page. When you use Below the Toolbar, the menu is spread horizontally across the the HTML page under the Toolbar area. When you use Left of Window, the menu is spread Vertically to the left of the <TABLE> representing the application's window.

ToolBar

This is the toolbar area at the top of the “window” in the HTML page (below the caption area).

Background color
You can specify the color to use for the Toolbar area. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button. If no color is specified here and you specified a Window background color in Window settings above, that color is used. If neither is specified and the application’s WINDOW has a COLOR attribute, that color is displayed in the browser. The template generated code calls the **`WebToolbar.SetBackground`** method to set this property.

Background image
You can specify an image to display as the background for the Toolbar area. Specify an image filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is no image. The template generated code calls the **`WebToolbar.SetBackground`** method to set this property.

Create extra close button
Specifies when to provide a Close button for a window. This button is in addition to any other buttons on the window. It is provided to replace the System Close button automatically provided by Windows but not automatically provided by a browser. If your windows all have close buttons, you do not need to provide an extra one. The choices are:

Never
Never creates an extra Close button.

If window has system menu and no visible buttons
Creates a Close button only when the WINDOW has a SYSTEM attribute and no other BUTTONs.

If window has system menu
Creates a Close button only when the WINDOW has a SYSTEM attribute

Always
Always creates a Close button.

Image for close
Specifies the icon to display for the Close button. Specify an icon filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is *EXIT.ICO*, a small blue X, (distributed with Clarion).

Client Area

This is the area of the “window” in the HTML page representing the application’s client area.

Background color
You can specify the color to use for your application’s client area. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button. If no color is specified here and you specified a Window background color in Window settings above, that color is used. If neither is specified and the application’s WINDOW has a COLOR attribute, that color is displayed in the browser. The template generated code calls the **`WebClientArea.SetBackground`** method to set this property.

Background image
You can specify an image to display as the background for your application’s client area. Specify an image filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is no image. The template generated code calls the WebClientArea.SetBackground method to set this property.

Tip

A background image tiles (i.e., it repeats as many times as its size allows) inside an HTML <TABLE> cell representing the application’s client area. Provide a small image that tiles to save bandwidth.

Class Overrides

Override default class
To override the IBC class, check this box and specify the Class Name, Header file (.INC), and Implementation file (.CLW) in the fields below.

Class Name
Specify the name of the class to use or the default class name if you wish to override the default class.

Header file
Specify a header file (the file containing the Class declarations) or select a file from a FILEDIALOG by pressing the ellipsis (...) button.

Implementation file
Specify an implementation file (the file containing the Class definitions or or source code) or select a file from a FILEDIALOG by pressing the ellipsis (...) button.

Internet Procedure Extension Template

This template allows you to customize the appearance and behavior of a procedure when it is executed over the Web. The settings you specify here are local in nature, that is they affect only this procedure. To change Global Settings: press the Global Button on the Application Generator, then press the Extensions button, and modify the settings for the Internet Application Extension.

To modify the settings, press the Internet Options button on the Procedure Properties window.

Note:

None of these settings affect the way your application works when running locally as a Windows executable.

Page Settings

When run over the Web, an application's window is displayed inside an HTML page (a Web page). The page settings allow you to specify a background color or background image for the HTML page. The template generated code calls the **`WebWindow.SetPageBackground`** method to set these properties.

Override Global settings
Check this box to override the Page settings in the Internet Application Global Extension template. Checking this box enables the other prompts.

Center Window on Page
Check this box to center the HTML representation of your window inside the Web page. This adds <CENTER></CENTER> HTML tags to the Web page.

Background color
You can specify the color to use for the Web page. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button. The default is no color (the equate is COLOR:NONE). This means that the browser's default page color is used.

Background image
You can specify an image to display as the background for the Web page. Specify an image filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is no image.

Window Settings

When run over the Web, an application’s window is represented by an HTML <TABLE>. The prompts on this tab allow you to specify the appearance of the “window” portion of the HTML page which displays when running the application over the Web.

Override Global settings
Check this box to override the Window settings in the Internet Application Global Extension template. Checking this box enables the other prompts.

Background color
You can specify the color to use for your application’s window. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button. The default is no color (the equate is COLOR:NONE), this means that the background color of the application’s window is used. The template generated code calls the `WebWindow.SetBackground` method to set this property.

Background image
You can specify an image to display as the background for your application’s window. Specify an image filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is no image. The template generated code calls the `WebWindow.SetBackground` method to set this property.

Tip

A background image tiles (i.e., it repeats as many times as its size allows) inside an HTML <TABLE> cell representing the application’s window. Provide a small image that tiles to save bandwidth.

Window border width
Specify the border width for your application’s window. The default is 2, which makes a thin border. Specify a 0 border width to display no border.

Help

Override Global settings
Check this box to override the Help settings in the Internet Application Global Extension template. Checking this box enables the other prompts.

URL of Help documents
The base location of the HTML files for your Help. For example, your HTML Help files are located in a separate subdirectory.

Help Window Style
You can optionally supply a style for your Help window

Help Ids are links within a base document
If your Help is designed as a single document with mid-page anchors, check this box. If not checked, the Help buttons reference individual HTML pages.

Help Document
The base document containing the mid-page anchors. This field is enabled only when the Help Ids are links within a base document box is checked.

Window Components
Press this button to specify settings to specify the appearance of the window's components (e.g., TOOLBAR, MENU, and Caption areas). These settings override any corresponding Global settings. See *Procedure Window Component Options*.

Return if launched from browser
Closes the procedure when executed over the Web. This effectively disables Web access to the procedure.

Controls

To Override Global settings:
Check the box to the left of an option to override the control settings in the Internet Application Global Extension template. Checking this box enables the prompt for that option.

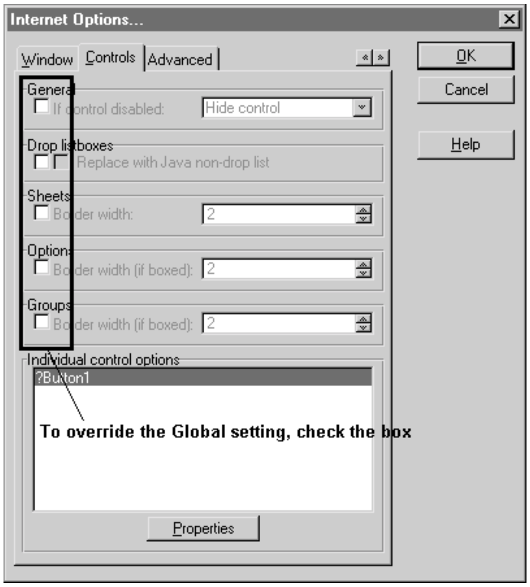
General

If control disabled
Specifies what to display on the browser when a window control is disabled. This option is provided because most HTML controls do not support disabling. This sets the `WebWindow.DisabledAction` property.The choices are:

Hide	Hides any disabled controls (the default).

Hide if cannot disable
Hides any disabled control when it cannot be disabled on the Web page. Most HTML controls cannot be disbled.

Show	Displays any disabled controls. It appears normally (i.e., it will appear to be enabled), but changes made to the control will not affect the underlying application.



General
☐ If control disabled: Hide control

Drop listboxes
☐ ☐ Replace with Java non-drop list

Sheets
☐ Border width: 2

Options
☐ Border width (if boxed): 2

Groups
☐ Border width (if boxed): 2

Individual control options
?Button1

To override the Global setting, check the box

Drop listboxes

Replace with Java non-drop list
This allows you to replace a drop-down list with a page-loaded Java Listbox. If your drop-down lists need to display more than one column, use this option.

Sheets

Border width
Specify the border width for SHEET controls. The default is 2, which makes a thin border. Specify a 0 border width to display no border. This sets the **`WebWindow.SheetBorderWidth`** property.

Options

Border width
Specify the border width for OPTION controls. This only applies to OPTIONs with the BOXED attribute. The default is 2 for a thin border. Specify a 0 border width to display no border. This sets the **`WebWindow.OptionBorderWidth`** property.

Groups

Border width
Specify the border width for GROUP controls. This only applies to GROUPs with the BOXED attribute. The default is 2, which makes a thin border. Specify a 0 border width to display no border. This sets the **`WebWindow.GroupBorderWidth`** property.

Individual Control Overrides

This section allows you to override the appearance or behavior of individual controls in the window. Highlight the control to modify and press the Properties button. See Individual Overrides for a Control.

MDI

This section determines the manner in which Application Menus and Toolbars are handled.

Tip

For control over specific Menu or Toolbar items, set the MDI overrides in the Frame Procedure's Internet Options.

Merge Frame Menu
Check this box to Merge the Frame's Menu when running this procedure.

Merge Frame Toolbar
Check this box to Merge the Frame's Toolbar when running this procedure.
For a Frame Procedure, you have additional options. See *Frame Procedure MDI Options*.

Advanced

Formatting

Override Global settings
Check this box to override the formatting settings in the Internet Application Global Extension template. Checking this box enables the other prompts.

Horizontal Pixels per Char
The number of pixels to consider for a character's width when calculating the size to create Java applets and Images.

Vertical Pixels per Char
The number of pixels to consider for a character's height when calculating the size to create Java applets and Images.

Delta for grid snapping
The number of pixels to consider before repositioning a control. Specify a value for X and a value for Y. Any time a control is within this range, it is not repositioned.

Note:

The numbers specified affect the overall appearance of the generated HTML page. For example, increasing the value of Vertical Pixels per Char will make the HTML Table cells taller.

Security

Transfer over a secure connection
If checked, data is transmitted using a Secure Socket Layer (SSL). This allows secure transactions on a procedure level. Keep in mind that encryption has a marked effect on performance. You should only enable security for procedures which transmit sensitive data.

This feature required installation of the secure version of the Application Broker. This feature is not available in this version.

Restrict Access to this procedure
Check this box to password protect the procedure and enable the two fields below.

Password
Specify a password or select a variable from the file schematic by pressing the ellipsis (...) button. A static password provides simple protection. For more information, see *Using Passwords*.

Case Sensitive
Check this box to enforce case sensitive validation of the password. If the box is not checked, case is ignored.

Window refresh

Show progress window
This controls the window associated with a Report or Process procedure. It is not available for other procedure types. Check this box to display the window associated with the Report Procedure when running over the Web. If not checked, the window is ignored. If the window in a Report Procedure contains a Pause Button control template, the box is checked and cannot be changed. In a Process procedure, the box is checked and cannot be changed. This makes sure the window displays.

Time between refresh
Specify the number of seconds between each refresh.

Action on Timer
Specify the action to perform when the timer event is reached. The choices are:

Partial Page refresh	Redisplays Java controls and HTML entry controls to reflect current data.
Submit page	Sends data to server application and redraws page as instructed by the server application
Complete Page refresh	Redraws the entire page.

Enable Refresh on timer
Check this box to refresh the entire page or only the page data based on a timer. A TIMER attribute on a WINDOW is independant of this setting. This setting is used on the Web and the TIMER attribute is used when the application runs under Windows.

Tip

This feature should be used sparingly to ensure minimal network traffic.

Time between refresh
Specify the number of seconds between each refresh.

Action on Timer
Specify the action to perform when the timer event is reached. The choices are:

Partial Page refresh	Redisplays Java controls and HTML entry controls to reflect current data.
Submit page	Sends data to server application and redraws page as instructed by the server application
Complete Page refresh	Redraws the entire page.

Individual Overrides for a Control

The prompts for individual control overrides change based on the type of control and its attributes. Every possible override is listed here with the conditional options noted.

Override Global settings
Check the box to the left of an option to override the control settings in the Internet Application Global Extension template. Checking this box enables the other prompts.

Display

If control disabled
Specifies what to display on the browser when a window control is disabled. This option is provided because most HTML controls do not support disabling. This sets the `IC:CurControl.DisabledAction` property. The choices are:

Hide	Hides any disabled controls (the default).
Hide if cannot disable	Hides any disabled control when it cannot be disabled on the Web page. Most HTML controls cannot be disbled.
Show	Displays any disabled controls. It appears normally (i.e., it will appear to be enabled), but changes made to the control will not affect the underlying application.

Hide if launched from browser
Check this box to hide the control when the application is run over the Web. This allows you to disable display of some data or remove some functionality for the Web version of your application without removing it from the Windows version.

Autospot Hyperlinks
This option is available for LIST and STRING controls. If checked, any data displayed which contains a valid hyperlink (i.e., those beginning with http:, https:, ftp:, mailto:, news:, telnet:, wais:, or gopher:) is made into a hyperlink jump.

Allow dynamic updates
This option is available for STRING controls. If checked, the string control is created on the HTML page as a Java string control which updates whenever a partial page update occurs.

Note:

STRING controls with a variable as the USE attribute automatically become Java String controls and do not need this override option. This is only appropriate for a static STRING which changes by a property assignment (e.g., ?String1{PROP:Text} = ‘New Text’).

Image Options

Update Image dynamically

This option is available for IMAGE controls. If checked, the control is created on the HTML page as a Java Image control which updates whenever a partial page update occurs.

Note:

IMAGE controls with a variable as the USE attribute automatically become Java Image controls and do not need this override option. This is only appropriate for a static IMAGE which changes by a property assignment (e.g., ?Image1{PROP:Text} = ‘New.gif’).

Alternative text

Optionally provide alternative text to display while the image is loading. This is added to the HTML IMG ALT= tag. Alternative text displays while the graphic file is transferred to browser (before the image displays) or instead of the image if the user disables image display in the browser’s preferences.

Border width

This option is available for SHEET, OPTION (if boxed) and GROUP (if boxed) controls. Specify the border width for the control. The default is 2, which makes a thin border. Specify a 0 border width to display no border.

HTML

One of the most powerful features of the IBC Templates is the ability to embed HTML code in the HTML pages which are output by the Web-enabled application. This feature allows you to add any HTML code at points before or after any control on the resulting Web page. This code does not affect the application when it is running as a Windows executable.

Using Embedded HTML, you can write any HTML code supported by the browser. You can insert your own custom JavaScript, Java applets, ActiveX controls, Shockwave files, or other objects.

Optionally, you can check the **Remove Default HTML generation** box to supress generation of HTML for the control.

See also: *Embedding HTML*.

Events

This tab allows you to override the default event handling for a control. This tab is only available for controls which generate events.

Every control has a default action. This determines how its events are processed. For example, a command button's default action is to submit the page to the server application and return a fresh Web page.

The ability to override the default event handling when the application is executed in a browser allows you to optimize the application for the Web environment and ensure that all of your embedded code is executed at the time you expect it to. For example, an entry control's events are processed on the browser by default. This means that any code on the Event:Accepted for an entry control is not executed until the page is submitted by a command button or other control that submits a page. Using Individual control overrides, you can specify a partial refresh on an Entry Control's Accepted event and embedded code executes as it would when running locally (under Windows).

By default, most controls which allow data entry have their events processed on the browser. This means your embedded code would not execute at the expected time (e.g., code in the Event:Accepted embed point for a control would not execute until the OK button submitted the page). This section allows you to override the Browser's event handling.

To override a control's event handling, highlight the event and press the Properties button.

Override default action
Check this to override the default action for the control event. Checking this box enables the other prompts.

Action on Event
Select the action to perform when the event occurs. The choices are:

Process on Browser	Allows event handling to be handled locally on the browser.
Partial page refresh	Specifies that all Java Controls and HTML Entry controls should receive and display updated data.
Complete page refresh	Replaces the entire page.

Classes

The Classes Tab lets you specify which classes (objects) the Templates use to accomplish various tasks, and the source modules that contain the class definitions. This approach gives you the capability to use as much of the IBC Library as you want and as much of your own classes as you want.

To change the class for an item or override the class, highlight it in the list, then press the Properties button.

Override default class
To override the IBC class, check this box and specify the Class Name, Header file, and Implementation file in the fields below.

Class Name
Specify the name of the class to use or the default class name if you wish to override the default class.

If you choose another class from the IBC Library, you do nto need to specify a Header or Implementation file.

Header file
Specify a header file (the file containing the Class declarations) or select a file from a FILEDIALOG by pressing the ellipsis (...) button.

Implementation file
Specify an implementation file (the file containing the Class definitions or or source code) or select a file from a FILEDIALOG by pressing the ellipsis (...) button.

Procedure Window Component Options

Caption

This is the area at the top of the “window” in the HTML page.

Override Global settings
Check this box to override the Caption settings in the Internet Application Global Extension template. Checking this box enables the other prompts.

Include caption
Check this box to display the Caption. If not checked, the Caption is not used.

Background color
You can specify the color to use for the Caption area. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button. The default is Navy Blue color (the equate is COLOR:Navy). If no color is specified and the application’s WINDOW has a COLOR attribute, that color is displayed in the browser. The template generated code calls the **`WebCaption.SetBackground`** method to set this property.

Background image
You can specify an image to display as the background for the Caption. Specify an image filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is no image. The template generated code calls the **`WebCaption.SetBackground`** method to set this property.

Tip

A background image tiles (i.e., it repeats as many times as its size allows) inside an HTML <TABLE> cell representing the application’s window caption area. Provide a small image that tiles to save bandwidth.

Alignment
You can control the alignment of the text in the caption area. The choices are *Left*, *Center*, or *Right* justification. The default is *Center*.

Font family name
This allows you to specify the typeface to display. Keep in mind that the browser can only display fonts which are installed on the client’s machine.

Font size
Optionally, specify the point size for the Font displayed in the caption Area. The default is no size specified, which uses the browser’s default font size.

Font color
You can specify the Font's color for the Caption area. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button.

Menu

This is the menu area at the top or side of the "window" in the HTML page.

Override Global settings
Check this box to override the Menu settings in the Internet Application Global Extension template. Checking this box enables the other prompts.

Background color
You can specify the color to use for the Menu area. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button. The template generated code calls the `WebMenubar.SetBackground` method to set this property.

Background image
You can specify an image to display as the background for the Menu area. Specify an image filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is no image. The template generated code calls the `WebMenubar.SetBackground` method to set this property.

Tip

A background image tiles (i.e., it repeats as many times as its size allows) inside an HTML <TABLE> cell representing the application's menu area. Provide a small image that tiles to save bandwidth.

Alignment
You can control the position of the menu alignment. The choices are *Above Toolbar* (the default) or *Left of Window*.

Toolbar

This is the toolbar area at the top of the "window" in the HTML page (below the caption area).

Override Global settings
Check this box to override the Toolbar settings in the Internet Application Global Extension template. Checking this box enables the other prompts.

Background color
You can specify the color to use for the Toolbar area. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button. The template generated code calls the `WebToolbar.SetBackground` method to set this property.

Background image
You can specify an image to display as the background for the Toolbar area. Specify an image filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is no image. The template generated code calls the `WebToolbar.SetBackground` method to set this property.

Tip

A background image tiles (i.e., it repeats as many times as its size allows) inside an HTML <TABLE> cell representing the application’s toolbar area. Provide a small image that tiles to save bandwidth.

Close button

Override Global settings
Check this box to override the Close button settings in the Internet Application Global Extension template. Checking this box enables the other prompts.

Create extra close button
Specifies when to provide a Close button for a window.

Image for close
Specify the icon to display for the Close button. Specify an icon filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is exit.ico (distributed with Clarion for Windows).

Client Area

This is the area of the “window” in the HTML page representing the application’s client area.

Override Global settings
Check this box to override the Client Area settings in the Internet Application Global Extension template. Checking this box enables the other prompts.

Background color
You can specify the color to use for the application’s client area. Specify a Color, a color equate, or select a color from the COLORDIALOG by pressing the ellipsis (...) button. The template generated code calls the `WebClientArea.SetBackground` method to set this property.

Background image
You can specify an image to display as the background for your application’s client area. Specify an image filename or select a file from a FILEDIALOG by pressing the ellipsis (...) button. The default is no image. The template generated code calls the `WebClientArea.SetBackground` method to set this property.

Tip

A background image tiles (i.e., it repeats as many times as its size allows) inside an HTML <TABLE> cell representing the application’s client area. Provide a small image that tiles to save bandwidth.

Frame Procedure MDI Options

Application Menu

Override Global settings
Check this box to override the Menu MDI settings in the Internet Application Global Extension template. Checking this box enables the other prompts.

Include on Child Windows
Select the option from the drop-down list. The choices are:

Global Setting	Menu choices appear on child windows as specified in the Global options.
All Menu Items	All menu choices appear on child windows.
No Menu Items	No menu choices appear on child windows.
Selected Menu Items	Allows you to select individual menu options from the list below.

Ignore code in frame's ACCEPT loop
Check this box to ignore any embedded code in the Application Frame's ACCEPT loop for menu items.

Application Toolbar

This section determines the manner in which Application Toolbar controls are handled. This allows you to specify which global Toolbar controls are displayed on “child” windows.

Override Global settings
Check this box to override the Toolbar MDI settings in the Internet Application Global Extension template. Checking this box enables the other prompts.

Include on Child Windows
Select the option from the drop-down list. The choices are:

Global Setting	Toolbar controls appear on child windows as specified in the Global options.
All Toolbar Items	All Toolbar items appear on child windows.
Standard Toolbar Only	Only the Standard Toolbar items appear on child windows.
No Toolbar Items	No Toolbar items appear on child windows.
Selected Toolbar Items	Allows you to select individual Toolbar items from the list below.

Ignore code in frame's ACCEPT loop
Check this box to ignore any embedded code in the Application Frame's ACCEPT loop for toolbar items.

Code Templates

Dynamic HTML Code Template

This code template allows you to insert dynamic HTML code in any of the Internet embed points. This template is only available for Embed points which can write to the delivered HTML page at runtime.

You can specify any valid Clarion expression in the entry box. Any variables used in the expression will use the current value at the time the HTML code is written.

Note:

When creating your expression to write HTML code, you must handle special characters, such as **<**, by using two characters in succession.

This template uses the **`Target.WriteLn`** method to write the value of the expression to the delivered HTML page.

See also: *Embedding HTML*

Static HTML Code Template

This code template allows you to insert static HTML code in any of the Internet embed points. This template is only available for Embed points which can write to the delivered HTML page at runtime.

You can specify any valid HTML code in the entry box.

This template uses the **`Target.WriteLn`** method to write the HTML code to the delivered HTML page.

Note:

If you use the Static HTML Code Template, special characters are handled automatically.

See also: *Embedding HTML*

GetCookie Code Template

This template allows you to retrieve a cookie from the client’s machine.

Cookie Name
Provide a name for the cookie. This is the name used in the SetCookie Code template to write the cookie. If the cookie does not exist, a null value is assigned to the Variable to Set.

Variable to Set
Select a variable from the file schematic by pressing the ellipsis (...) button. The value of the cookie is assigned to the variable.

See also: *SetCookie Code Template*, *Cookies (Persistent Client Data)*

SetCookie Code Template

This template allows you to set a cookie on the client’s machine for later retrieval.

Cookie Name
Provide a name for the cookie. This is the name to use in the GetCookie Code template to retrieve the cookie. If a cookie of the same name exists, it is overwritten.

New Value
Specify a value or select a variable from the file schematic by pressing the ellipsis (...) button. This value is assigned to the cookie.

See also: *GetCookie Code Template*, *Cookies (Persistent Client Data)*

Cookies (Persistent Client Data)

Cookies are a method for Web servers to both store and retrieve information on the client side of the connection. This allows a server to store data on the client’s machine and retrieve it later.

A server can send a piece of data to the client (browser) which the client stores locally. This is known as a cookie (the name has no known origin). Cookies contain a range of URLs for which it is valid. Later, when the client returns to a URL within that range, the server can query the cookie and use that data. A server cannot retrieve information from other servers (i.e., a server cannot query a cookie that is out of its domain range).

This mechanism is similar to the INI file storage and retrieval paradigm in Windows (GETINI and PUTINI) and provides a method for identifying user preferences, and other data. For example, an application which requires a user to provide their name before entering can use a cookie to avoid the Login process after the first visit.

Note:

Cookies are machine specific so a client who accesses a site from more than one machine will need to provide the cookie information once for each machine so a cookie is stored on the machine. In addition, cookies are browser specific, so a client who uses more than one browser, will need to set and get cookies for each browser.

Your Web-enabled applications can use cookies to store user preferences such as the default city and state for new records. These settings can be retrieved the next time the user runs the application over the Web.

See also: *GetCookie Code Template*, *SetCookie Code Template*

AddServerProperty Code Template

This template allows you to set the value of the specified outgoing http item in the HTTP header.

Property Name
Provide the property name to set.

Property Value
Select a variable from the file schematic by pressing the ellipsis (...) button. The value of the variable is assigned to the property.

See Also : *GetServerProperty Code Template*

GetServerProperty Code Template

This template allows you to get the value of the specified http item in the HTTP header.

Property Name
Provide a name for the HTTP property. If the HTTP field does not exist, a null value is assigned to the Variable to Set.

Variable to Set
Select a variable from the file schematic by pressing the ellipsis (...) button. The value of the property is assigned to the variable.

See Also : *SetServerProperty Code Template*

11 - Web Application Design Considerations

Most common Windows application design rules apply to Web application design. It is equally important to provide a consistent, understandable interface under either platform.

Keep in mind that the Web “platform” is not Windows. Your interface should be intuitive for users on all supported platforms. The Java controls in the Java Support Library are intuitive, but you may want to provide a brief explanation of how they work in your application to facilitate their use.

Bandwidth Usage Considerations

The web introduces one additional programming challenge—bandwidth conservation. It is important to keep your windows simple and utilize all the methods available to reduce the amount of network traffic. This section provides some pointers, but is by no means complete. It is intended to give you food for thought while designing applications.

Use Partial Refresh whenever possible

The use of a Partial Refresh, where appropriate, is the best way to optimize your Web applications.

There are many times when a partial refresh is appropriate but a full refresh is the default. This is necessary because the templates cannot anticipate every possibility. For example, a multi-sorted list which has no controls populated on the Tabs performs better if you use Individual Control Overrides to specify a Partial refresh when a tab is selected. This will only change the data in the listbox instead of replacing the entire page.

To override a SHEETs behavior for the example above, follow these steps:

1. From the *Procedure Properties* window, press the **Internet Options** button.
2. Select the **Controls** Tab.
3. Highlight the Sheet control in the **Individual Control Options** list (the wizard generated SHEETs are usually called ?CurrentTab).
4. Press the **Properties** button, then select the **Events** tab.
5. Highlight the *Accepted* event, then press the **Properties** button.
6. Check the **Override default action** box, then select *Partial page refresh* from the drop-down list.

7. Press the **OK** buttons on all the windows to save and exit.

One other aspect of Partial Refresh is its use to Update Controls over the Web. In Windows applications, programmers often embed code to update one control when the value of another control changes. For example, you might embed code to change the total of a line item when the quantity of items changes. The Webtree tutorial application has code like this in the UpdateItems procedure. The embedded code is tied to the EVENT:Accepted on each control. In other words, when the user changes the value in a control and tabs off it or selects another control with a mouse click, the code is executed.

When an application runs over the Web, ENTRY controls are processed on the browser by default. In other words, there is no interaction between the browser and the server application—unless you change the event handling options for that control. If you want to update controls over the Web, change the action for controls to ensure that embedded code is executed on the Event:Accepted.

Be frugal with controls

Populate as few controls as necessary on a window. This is good practice in Windows application design and is even more important in a browser/server implementation

When using listboxes, populate as few controls in the list as needed to uniquely identify a record for a user. This reduces the amount of data sent to fill the list. If you want to display more data for each record, you can populate hotfields next to the listbox and they will update as the user scrolls.

Use graphics sparingly

This is a common rule for web design. You should limit the number of graphics to ensure rapid page loading. In addition, you should reduce the file size as much as possible to further save bandwidth usage. Many graphics utilities have tools to adjust graphics files for web usage.

Covering the Download with a Splash Window

In order for a browser to “run” a Web-enabled application, the Java Support Library (JSL) must be available to the client browser. First-time users must download either Clarion.CAB (for Microsoft Internet Explorer) or Clarion.ZIP (for Netscape). In most browsers, the JSL is only downloaded once and remains cached (until the user clears that cache). Although the JSL is very compact for the degree of functionality it provides, it can still take several minutes to download over a 28.8 modem. With that in mind, you may want to use a “splash screen” window to alert first-time users that the download is in progress. By placing a Java Button on that window, you can prevent users from continuing until the JSL is downloaded and the Java button is initialized.

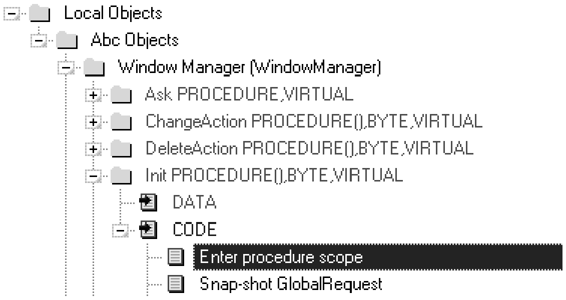
Create the Window and Change the BUTTON to a Java Button

Create a procedure using the Window Procedure template. These instructions assume you have named your procedure-*Splash*. This window should contains some text and a Close Button control template. You can change the text on the BUTTON to **Continue**. Since the button is created as an HTML button by default, you should specify that you want it to be a Java button so that it will not be available until the JSL has downloaded.

1. In the Application Tree, highlight the new procedure, then press the **Properties** icon button.
2. Press the **Internet Options** button.
3. Select the **Controls** tab.
4. Highlight the close button control template (the default name is *?Close*) in the **Individual Control Options** list, then press the **Properties** button.
5. Select the **Classes** tab.
6. Check the **Override default Class** box, then select the *WebJavaButtonClass* from the **Class Name** drop-down list.
7. Press the **OK** button.

Call the procedure before opening the Application Frame

1. In the Application Tree, highlight the *Main* procedure, then press the **Properties** icon button. This opens the *Procedure Properties* window.
2. Press the **Embeds** button. This opens the *Embedded Source* window.
3. Highlight the embed point as shown below:



- Local Objects
  - Abc Objects
    - Window Manager (WindowManager)
      - Ask PROCEDURE,VIRTUAL
      - ChangeAction PROCEDURE(),BYTE,VIRTUAL
      - DeleteAction PROCEDURE(),BYTE,VIRTUAL
      - Init PROCEDURE(),BYTE,VIRTUAL
        - DATA
        - CODE
          - Enter procedure scope
          - Snap-shot GlobalRequest

4. Press the **Insert** button. This opens the *Select Embed Type* window.

5. Highlight *Source*, then press the **Select** button.

6. In the *Embedded Source* editor, type the following source code:

   ```
   IF WebServer.Active THEN Splash.
   ```

   This makes sure that the Splash procedure is only called when the application is running over the Web.

7. Make sure this embed is listed before the call to any other procedure using the up or down button.

   This ensures that the Splash procedure is called before any other window opens.

8. Press the **Close** button on the *Embedded Source* window and the **OK** button on the *Procedure Properties* window.

9. Press the **Procedures** button.This opens the *Called Procedures* window.

10. Highlight *Splash*, then press the **OK** button.

    This connects the *Splash* procedure to the *Main* procedure in the Application Tree. This is necessary if your application is using Local MAPs.

Cosmetic Design Considerations

Using Groups

When you populate a GROUP on a WINDOW, control declaration statements do not necessarily end up inside the GROUP structure. This may cause an HTML representation that does not look like the original window. Make sure the controls you want inside the GROUP are actually inside the GROUP structure.

In the first example below (Badwind), the control declaration statements are all outside the GROUP structure. This window displays fine in Windows because the AT attribute values control the position and size of the GROUP box. When running over the Web, the GROUP box is an HTML <TABLE> cell and is controlled by its contents.

```
Badwind  WINDOW('Caption'),AT(,,260,120),GRAY
           GROUP('Customer Info'),AT(5,9,205,102),USE(?Group1),BOXED
           END
           PROMPT('Customer:'),AT(11,28),USE(?CUST:Name:Prompt)
           ENTRY(@s30),AT(61,26)USE(CUST:Name),LEFT,REQ
           PROMPT('Address:'),AT(15,47),USE(?CUST:Address:Prompt)
           ENTRY(@s30),AT(61,45),USE(CUST:Address),LEFT
           PROMPT('City:'),AT(29,69),USE(?CUST:City:Prompt)
           ENTRY(@s20),AT(61,67),USE(CUST:City),INS
           PROMPT('State:'),AT(25,88),USE(?CUST:State:Prompt)
           ENTRY(@s2),AT(61,86),USE(CUST:State),LEFT,UPR
         END
```

In the second example (Goodwind), the control declaration statements are within the GROUP structure (i.e., between the GROUP and END statements) and will display as expected when run over the Web.

```
Goodwind WINDOW('Caption'),AT(,,260,120),GRAY
          GROUP('Customer Info'),AT(5,9,205,102),USE(?Group1),BOXED
           PROMPT('Customer:'),AT(11,28),USE(?CUST:Name:Prompt)
           ENTRY(@s30),AT(61,26)USE(CUST:Name),LEFT,REQ
           PROMPT('Address:'),AT(15,47),USE(?CUST:Address:Prompt)
           ENTRY(@s30),AT(61,45),USE(CUST:Address),LEFT
           PROMPT('City:'),AT(29,69),USE(?CUST:City:Prompt)
           ENTRY(@s20),AT(61,67),USE(CUST:City),INS
           PROMPT('State:'),AT(25,88),USE(?CUST:State:Prompt)
           ENTRY(@s2),AT(61,86),USE(CUST:State),LEFT,UPR
          END
         END
```

Using Images

Java Image controls update automatically when the value of its source variable changes (i.e., whenever a partial page update occurs). To use this feature for an IMAGE which changes by a property assignment (e.g., ?Image1{PROP:Text} = ‘New.gif’), use Individual Control Overrides for the Image Control and specify to update dynamically.

Graphic files used by IMAGE controls are extracted to the temporary runtime directory for the connection unless they are found in the /PUBLIC directory. The runtime library will extract files of various types, but most browsers only support GIF and JPG formats. Therefore, you should limit the graphic formats of IMAGE controls in a web-enabled application to those two types. You could also choose to hide an IMAGE which uses a format not supported by browsers using Individual Control Overrides. If an IMAGE is based on a file that is not linked in, you should deploy the image file to the application's directory.

You should provide alternative text for images (in Individual Control Overrides). This is added to the HTML <IMG ALT=> tag. Alternative text displays while the graphic file is transferred to browser (before the image displays) or instead of the image if the user disables image display in the browsers preferences.

Icons used in LIST controls or on BUTTONs are not automatically extracted and should be deployed to the /PUBLIC directory.

If you are referencing an image in HTML code, you must indicate the location of the image file. If you are deploying under the EXE version of the Application Broker you can prefix the filename with a leading forward slash and deploy the image to the /PUBLIC directory. For example <IMG SRC="/LOGO.GIF">. If you are using the ISAPI DLL version of the Application Broker, you must use the **`SELF.FILES.GETAlias`** method to determine the virtual path to the file. For example:

```
Target.WriteLN('<<IMG SRC="' & SELF.Files.GetAlias('mygif.gif')& '">')
```

would find the mygif.gif file in any directory exposed to the server application.

User Interface Design Considerations

MDI window access

Windows applications often use a Multiple Document Interface (MDI). This allows several instances of an MDI child window to open. Each of these Child windows is available and can receive focus using several navigation methods (e.g., the Window menu). This is very convenient, but has some implications when porting the application to the Web platform. A web page in a browser is a single document, however, the underlying server application can be an MDI application and allow multiple windows. Many windows could be open on the server application, but the browser only displays the current window. You should keep this in mind when designing your application.

In a Web-enabled application, you can allow all menu and toolbar command to be visible on child windows. This can be useful to allow a user to enter different areas of the application without closing a child window to get to the main menu or toolbar. This also has the potential pitfall of allowing a user to open multiple instances of a procedure. Although only one will be visible at a time, there could be several windows open. If there are two or more of the same window open, it may appear to the user that the procedure did not close when the Close button was pressed. For this reason, you should either restrict access to the Global Menu/toolbar or limit each MDI procedure to a single instance using Thread limiting code. One technique of limiting threads is demonstrated in one of the standard Clarion Examples—EventMgr.APP.

Restricting Edit-In-Place

The ABC Templates in Clarion allow you to enable Edit-In-Place with a single checkbox. This feature, however, is not supported when running over the Web. Over the Web, you must have a separate Form for updates. There is a simple method to alternate between edit-in-place when running locally in Windows and calling a form when running over the Web.

If you enable Edit-In-Place and specify an update procedure with the BrowseBox control template, you have two-thirds of your work done. The template generated code either calls a separate update procedure or does edit-in-place depending on the value of the BRWn.AskProcedure property. Set the **`BRWn.AskProcedure`** property to 0 (zero) and you have Edit-in-Place; Set it to 1 (One) and you call the update procedure.

To use Edit-in-place for local Windows and a form when running over the Web:

1. Select the Browse procedure, then press the **Properties** icon button.

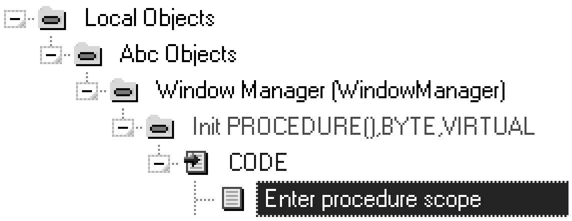
2. In the UpdateButton section of the *Procedure Properties* window, check the **Use Edit in Place** box.

   Notice that an update procedure is already specified. Make sure to leave that procedure named.

Next, embed the code to set the update action to call Edit-in-Place when running in Windows and call the form when running over the Web.

3. Press the **Embeds** button. This opens the Embedded Source window.

4. Highlight the embed point as shown below then press the **Insert** button.



- Local Objects
  - Abc Objects
    - Window Manager (WindowManager)
      - Init PROCEDURE(),BYTE,VIRTUAL
        - CODE
          - Enter procedure scope

5. Highlight *Source*, then press the **Select** button.

6. In the Embedded Source editor, type the following source code:

```
IF WebServer.Active
 BRW1:AskProcedure = 1
END
```

7. Exit the Source editor and save the changes.

8. Press the **Close** button on the *Embedded Source* window.

Unsupported Windows Standard Dialogs

There are certain Windows standard dialogs which are not appropriate for an application running over the Web. Calling any of these will display a *Not Supported* Message:

COLORDIALOG
FILEDIALOG
FONTDIALOG
PRINTERDIALOG

If you are calling any of these with a BUTTON control, use the Individual Control Options to "Hide if launched from Browser." (Internet Options Controls).

If you are calling the function in source code, enclose the function call inside a conditional structure. For example:

```
IF NOT WebServer.Active      ! Check if running over the web
  retval=COLORDIALOG()       ! if not, call the colordialog
END
```

Using Command Line Parameters

If your application needs to receive command line parameters, you can pass them on the browser's command line or via a hyperlink.

On the browser's location (URL) entry, specify the URL followed by the executable name, followed by the dot zero (.0) followed by a question mark and the parameter. For example,

```
HTTP://mydomain.com/myapp.exe.0?MyParameter
```

To handle the parameter in your application, you must interrogate the WebServer.CommandLine property. If you are creating a hybrid application and want to receive command line parameters from either Windows or the Web, use code similar to the example below:

```
IF WebServer.Active                     !Check if running over the web
  PRE:MyField = WebServer.CommandLine  !assign value to variable
ELSE                                    !if it is running locally
  PRE:MyField = COMMAND('')             !assign value to variable
END
```

Note:

If you are passing multiple parameters, you must parse the string to access the individual parameters.

Changing the Class for an individual control

At times, you may want to change a single control to use a different class than the default. For example, a STRING control that displays a variable defaults to a Java String control and you may want it to be plain HTML text. You can change this on a control-by-control basis on the Individual Control Overrides Classes Tab. In this example, you are not actually overriding the class, but merely specifying a different class to use for the control.

1. From the *Procedure Properties* window, press the **Internet Options** button.
2. Select the **Controls** tab.
3. Highlight the control in the **Individual Control Options** list, then press the **Properties** button.
4. Select the **Classes** tab, and check the **Override Default Class** box.
5. Select the class to use from the drop-down list. You do not need to provide the Header File and Implementation files.

You can use the same technique to change a JavaImageControl to an HTML <IMG> control.

API calls

Windows API calls are tied to the machine on which an application is running. Web-enabled applications are actually running on the server machine and a representation is sent to the client in the form of HTML pages. Therefore, any API calls in your application execute on the server machine.

In many cases, this will not be appropriate. For example, playing a sound file on a server is generally not a good idea and the user running the application won't hear it. In those cases, you should inhibit the call when the application is running over the web.

If you are making the call with a BUTTON control, use the Individual Control Options to "Hide if launched from Browser." (Internet Options Controls).

If you are making the call in source code, enclose the function call inside a conditional structure. For example:

```
IF NOT WebServer.Active      ! Check if running over the web
  SoundFile='fanfare.wav'
 sndPlaySound(SoundFile,1)
END
```

In other cases, it will be appropriate to make the call on the server. For example, a procedure which uses MAPI to send email from the server based on an event. In those cases, you should make sure the call works properly on the server. It should behave the same way when executed over the web.

In a similar manner, reports without Print Preview enabled will print on the server. This may be appropriate in some cases, but it is important to understand its behavior.

Security Considerations

There are two methods of implementing security in your web applications.

- Implementing security into the underlying application.
- Restricting access (Password protecting) a procedure when it is started over the Web.

The first method—implementing security into the original application—requires no additional consideration in your Web application. The original security enforcement in the Windows version should work the same way in your Web application.

The second method—restricting access when running over the Web—uses the browser's built-in authentication.

Using Passwords

The Internet Procedure Extension template's Password protection uses the browser's built-in HTTP authentication support. When a password protected procedure is called, the browser's authentication window displays. You do not need to create a window to collect login information. Password protection is based on an area, a username and a password. The area is the protected procedure.

When a browser requests a password protected area, it gets a response back requesting the username and password for the area. By default, the area name is created from the title of the window, and the name of the procedure. This is stored in the **`WebWindow.AuthorizeArea`** property. The browser prompts the user for a user name, and a password. These are then sent to the program for validation. If the program accepts the password (i.e., it RETURNs TRUE from the WebWindow.ValidatePassword method), the new page is displayed, otherwise the browser prompts again. After three attempts the browser displays a message informing the user that access is denied. This page automatically returns the user to the last active place in the program.

Note:

If the page has already been visited in the current session the browser will normally supply the user name and the password without prompting the user. This feature is built-in to most browsers.

Two levels of password support are built into the procedure template. The simplest method is to select restricted access and specify a single password or a variable. This is automatically checked by the template, and ignores the username. If you use a variable, it compares the password entered with the variable's current value.

The more advanced method is to override the **`WebWindow.ValidatePassword`** method by entering code into the *Internet- Password Validation Code Section* embed point. This embed point is inside a method with two parameters: UserName and Password, which it receives from the browser. You should return TRUE if the password is valid, and FALSE if it is not valid. This allows you to look up the information in a file, or use any other method you choose to validate the password.

Example:

```
USE:UserID = UserName
IF Access:UserList.Fetch(USE:UserIDKEY)
  RETURN(False)
END
IF USE:UserPassword = Password
  RETURN(True)
Else
  RETURN(False)
END
```

Optionally, you can change the message displayed on the browser's password dialog by assigning a value to **`WebWindow.AuthorizeArea`** in the *Internet-After Initializing the window object* embed point.

Using Embedded HTML

One of the most powerful features of the IBC Templates is the ability to embed HTML code in the HTML pages which are output by the web-enabled application running via the Application Broker. When you embed HTML code (using the special embed points added by the templates), it is inserted at the specified location in the HTML file returned to the browser which executed the application.

There are two methods for embedding HTML:

1. In the Internet Procedure Extension Template, Individual Control Overrides. This provides two text entry controls into which you write HTML code.
2. Using the Dynamic HTML Code Template or the Static HTML Code Template in one of the Internet embed points. These templates use the virtual method Target.WriteLn to write to the delivered HTML file at runtime. The Static HTML code template allows you to embed HTML code exactly as written. The Dynamic HTML template allows you to combine HTML code with variables from your application.

Optionally, you can use the **`Target.WriteLn`** method yourself in embedded source code in any of the appropriate embed points.

These Embed points are identified by INTERNET at the beginning of the description. Using the **`Target.WriteLn`** method in one of these embed points allows you to add any HTML code at various points in the HTML document delivered to the user at runtime. This code does not affect the application when it is running as a Windows program.

For example, if you want a block of text to appear on the bottom of the page delivered by the Application Broker for a procedure in your application, you would insert the Static HTML Code Template at the *Internet, before the closing </BODY> tag* embed point in the Application Generator and specify the HTML code. This HTML code is added to the resulting HTML page delivered to a browser client.

You can use the virtual method **`Target.WriteLn`** in any the embed points where the Dynamic HTML Code Template and the Static HTML Code Template are available.

Example:

Insert this code in the *Internet, before the closing </BODY> tag* embed:

```
Target.WriteLn('<<p>Copyright 2000, SoftVelocity&trade; Incorporated, All
Rights Reserved.<</p>')
```

Note:

When hand-coding Clarion source to write HTML code, remember to handle special characters, such as <, by using two characters in succession. If you use the Static HTML Code Template, this is handled automatically.

One benefit of using Clarion code in these embed points is the ability to control the HTML code you want to write. The example below shows a simple method of displaying a random hyperlink:

```
EXECUTE RANDOM(1,5)
 Target.WriteLn('<<A HREF="http://www.softvelocity.com">Visit SoftVelocity<</A>')
 Target.WriteLn('<<A HREF="http://www.clariononline.com">Visit ClarionOnline<</A>')
 Target.WriteLn('<<A HREF="http://www.icetips.com">Visit IceTips<</A>')
 Target.WriteLn('<<A HREF="http://www.finatics.com">Visit the Finatics<</A>')
 Target.WriteLn('<<A HREF="http://www.softvelocity.com/news.htm">SoftVelocity News<</A>')
END
```

Using references to files in embedded HTML code

When using references to files in embedded HTML code, remember that each session has its own temporary directory. Therefore, /PUBLIC is never the current directory for delivered web pages. This means that you must reference the location of files. There are two ways to do this.

If you are referencing an image in HTML code, you must indicate the location of the image file. If you are deploying under the EXE version of the Application Broker you can prefix the filename with a leading forward slash and deploy the image to the /PUBLIC directory. For example <IMG SRC="/LOGO.GIF">.

If you are using the ISAPI DLL version of the Application Broker, you must use the SELF.FILES.GetAlias() method to determine the virtual path to the file.

For example:

```
Target.WriteLN('<<IMG SRC="' & SELF.Files.GetAlias('mygif.gif') & '">')
```

would find the mygif.gif file in any directory exposed to the server application.

Note:

The preferred method is to use the SELF.Files.GetAlias() method because it works under both the ISAPI DLL and the EXE version of the application broker.

To use your own Java Applet class files, use the CODEBASE= tag as shown below.

If you are deploying under the EXE version of the Application Broker you can reference the <CODEBASE> as a leading forward slash and deploy the .CLASS file to the /PUBLIC directory. If you are using the ISAPI DLL version of the Application Broker, you must use the SELF.FILES.GetAlias() method to determine the virtual path to use for the <CODEBASE>.

Embedded HTML Examples:

```
!  HTML code

<IMG SRC=”/mypic.gif”>
<applet codebase=”/” code=”TickerTape.class” width=”500" height=”32">
</applet>

!  Embedded Source Examples (in any Internet Embed Point)

Target.WriteLN(‘<<IMG SRC=”’ & SELF.FILES.GETAlias(‘mygif.gif’)& ‘“>’)
Target.Writeln(‘<<applet ‘)
Target.Writeln(‘Codebase = “‘ & SELF.FILES.GETAlias() & ‘“ ‘)
Target.Writeln(‘code=”TickerTape.class”>’)
Target.Writeln(‘<</applet>’)
```

Note:

In an APPLET HTMLtag, the CODEBASE attribute must precede the code attribute. This is listed in the wrong order in some HTML references. HTML code with the attributes in the wrong order can cause the applet to fail (due to a "Not Found" error).

Implementing Help in your Web Application

References are made to HTML pages based on the current window's Help ID. This is constructed in one of two ways: Using a Base Document with Mid-Page anchors, or Using individual help Documents. This is specified in the Global Application Extension Template or in the Procedure Extension template's Internet Options.

Using a Base Document with Mid-Page anchors

This method uses a single web page with mid-page bookmarks or anchors. The call to the page is constructed by appending the Help ID to the base page name with a # symbol between them (e.g., HELP.HTM#IDNAME). Clicking on the Help button causes the page to open and scroll to the appropriate anchor. In the example below, the first window has a HelpID of ~FirstWindowID. This means that the Help button will call HelpFile.HTM#FirstWindowID.

Example:

```
<html>
<head>
<title>Example Help Document</title>
</head>
<body background=”bgrnd.gif” bgcolor=”#FFFFFF”>
<h1 align=”center”>Program Help </h1>
Introductory Text......
Introductory Text......
Introductory Text......
Introductory Text......
<h2 align=”center”><a name=”FirstWindowID”>Help For First Window</a></h2>
Explanation of how this procedure works
Explanation of how this procedure works
Explanation of how this procedure works
Explanation of how this procedure works
<h2 align=”center”><a name=”SecondWindowID”>Help For Second
Window</a></h2>
Explanation of how this procedure works
Explanation of how this procedure works
Explanation of how this procedure works
Explanation of how this procedure works
</body>
</html>
```

Using individual help Documents

This method uses a single web page for each window. The call to the page is constructed by prepending the Help ID to .HTM extension. Clicking on the Help button causes the page to open.

Both methods open the page in a new browser window named “_HELP”. If you open your application inside a frame set where one of the frames is named “_HELP”, the help page opens in that frame.

A web-enabled application executed by the Application Broker creates HTML files in the /PUBLIC directory. These pages are sent to the browser which started the application and refreshed and re-sent when the client interacts with the application.

Windows Controls and their HTML Equivalents

A web-enabled application executed by the Application Broker delivers HTML to the browser which started the application and refreshed and re-sent as the user interacts with the Web page representing the application.

Certain controls translate easily to HTML, while others are created as JAVA classes using the Clarion Java Support Library. Certain windows controls have not been fully implemented in this release.

The list below shows the standard windows controls supported by Clarion and the equivalent created by an Internet Connect web-enabled application.

STRING (a variable string)
Displays as a Java String Control, which updates dynamically.

STRING (a text string)
Displays as text by default. By setting individual control overrides, it can display as a Java String Control, which updates dynamically. If you are updating the STRING in your application using a property assignment, you should specify that the string update dynamically.

IMAGE
A static image displays as an HTML image <IMG> with its source specified as the graphic file in your application. By setting individual control overrides, it can display as a Java Image Control, which updates dynamically.

REGION
Partial support. A REGION that covers an IMAGE control and has functionality implemented in its EVENT:Accepted creates the HTML image as an image map (USEMAP=) with the functionality of the region associated with that portion of the image.

LINE
Not supported--use Embedded HTML to display a Horizontal Rule <HR> or an image <IMG SRC="file.gif">.

BOX
Not supported--use Embedded HTML to display an image <IMG SRC="file.gif">.

ELLIPSE
Not supported--use Embedded HTML to display an image <IMG SRC="file.gif">.

ENTRY
Created as an HTML entry field <INPUT TYPE=TEXT VALUE = value in field >. Entry patterns are not supported.

BUTTON
Created as an <INPUT TYPE=SUBMIT > unless it has an ICON, then a Java button is created which displays the Icon. Icons displayed on Java buttons must be deployed to the /PUBLIC directory.

PROMPT
Displays as text.

OPTION
Created as an HTML <OPTION>. If an OPTION has the BOXED attribute, then it is implemented in HTML as a <TABLE> with the border specified in the Global or Procedure options for OPTIONs.

CHECK
Created as an HTML checkbox <INPUT TYPE=CHECKBOX VALUE = value in field >

GROUP
If a GROUP has the BOXED attribute, then it is implemented in HTML as a <TABLE> with the border specified in the Global or Procedure options for GROUPs.

LIST
Creates a Java Listbox which supports most of the LIST attributes, including conditional colors and icons. Icons must be deployed to the /PUBLIC directory. When the Java Listbox has focus in the browser, the navigation keys are supported (arrow-up, page-up, etc.). If the LIST has a locator, the Java Listbox supports it when it has focus. Double-click handling is also supported. Drag-and-drop, edit-in-place, and right-click popups are not supported.

Tree Creates a Java Tree Listbox. Supports all attributes, including conditional colors and icons. Icons must be deployed to the /PUBLIC directory.

FileDropCombo
Created as an HTML drop-down (<SELECT> structure) with the values from the file created as Options. This does not support multiple columns. Optionally, you can create as a Java Non-drop list which supports multiple columns.

DropList
Created as an HTML drop-down (<SELECT> structure). This does not support multiple columns. Optionally, you can create as a Java Non-drop list which supports multiple columns.

DropCombo
Created as an HTML entry field <INPUT TYPE=TEXT VALUE = value in field >

COMBO
Created as an HTML entry field <INPUT TYPE=TEXT VALUE = value in field >.

SPIN
Created as an HTML entry field <INPUT TYPE=TEXT VALUE = value in field >.

TEXT
Created as an HTML Text field <TEXTAREA >.

CUSTOM (.VBX)
Not supported.

MENU
Creates a list of hyperlinks which display across the top of the HTML page or to the left of the window, as specified in the Global Internet Options.

ITEM
See **MENU**.

RADIO
Creates an HTML Radio button.

APPLICATION
HTML <TABLE> inside an HTML page.

WINDOW
HTML <TABLE> inside an HTML page.

REPORT
If Print Preview is enabled, this creates a series of HTML pages with Java navigation buttons (Next page, Previous page, etc.). If Preview is not enabled, the report will print on the server.

HEADER, FOOTER, BREAK, FORM, DETAIL
See REPORT.

OLE
Not Supported (except via embedding an ActiveX in Embedded HTML).

PROGRESS
Not supported.

SHEET
Created as JAVA Tab controls.

TAB
Created as JAVA Tab controls.

PANEL
Not supported. You may use a GROUP with the appropriate borderwidth to provide a similar appearance.

TOOLBAR
Created as a row in an HTML <TABLE>. Controls on the toolbar are placed as specified in the Global or procedure Internet Options.

Hand Coded Applications

About This Section

The Internet Connect Templates generate the code necessary to Web-enable Clarion applications. However, you do not have to use the Internet Connect Templates to Web-enable your programs.

That is, you can use the IBC Library to Web-enable your hand coded programs. This chapter presents a minimal “Hello Web” hand coded program that uses the IBC Library. This chapter also discusses the IBC Library’s project system requirements.

The easiest way to learn to use the IBC Library within hand coded programs is to Web-enable an application with the Internet Connect Templates, then study the template generated code.

HelloWeb Example Program

The following hybrid Web/Windows program displays a single window or Web page with a “Hello Web” message and a “Goodbye Web” button to shut down the program.

```
HelloWeb   PROGRAM
LinkBaseClasses   EQUATE(1)                  !Enable LINK on CLASS declarations
                                             !so linker can find implementation
                                             !(.clw) files
BaseClassDllMode  EQUATE(0)                  !Activate DLL on CLASS declarations
                                             !for required 32-bit dereference
 INCLUDE('ICBROKER.INC')                     !Declare BrokerClass
 INCLUDE('ICWINDOW.INC')                     !Declare WebWindowClass
 INCLUDE('ICSTD.EQU')                        !Declare IC standard EQUATEs
 MAP
  Hello                                      !Prototype Hello procedure
  WebControlFactory(SIGNED),*WebControlClass  !Prototype WebControlFactory
     MODULE('')
       SetWebActiveFrame(<*WebFrameClass>)   !Prototype SetWebActiveFrame
     END
   END
Broker          BrokerClass                  !Declare Broker object
HtmlManager     HtmlClass                    !Declare HtmlManager object
JavaEvents      JslEventsClass               !Declare JavaEvents object
WebServer       WebServerClass               !Declare WebServer object
WebFilesManager WebFilesClass                !Declare WebFilesManager object
ICServerWin     WINDOW,AT(-1,-1,0,0)         !Declare “invisible” server window
                END
  CODE
  SetWebActiveFrame()                        !Tell IBC objects (WebWindow) there
                                             !is no active APPLICATION frame
  WebFilesManager.Init(1, '')                !Initialize WebFilesManager
  JavaEvents.Init                            !Initialize JavaEvents
  Broker.Init('HelloWeb', WebFilesManager)   !Initialize Broker
  HtmlManager.Init(WebFilesManager)            !Initialize HtmlManager
  WebServer.Init(Broker,'',600,'',WebFilesManager) !Initialize WebServer
  IF (WebServer.GetInternetEnabled())          !If launched by Application Broker
    OPEN(ICServerWin)                          ! open “invisible” window on server
    ACCEPT
      IF (EVENT() = EVENT:OpenWindow)
        WebServer.Connect                      !Establish channel to App Broker
        Hello                                  !Call Hello (Web mode)
        BREAK
      END
    END
  ELSE                                         !If not launched by App Broker
    Hello                                      ! call Hello (Windows mode)
  END
  WebServer.Kill                               !Shut down WebServer object
  HtmlManager.Kill                             !Shut down HtmlManager object
  Broker.Kill()                                !Shut down Broker object
  JavaEvents.Kill                              !Shut down JavaEvents object
  WebFilesManager.Kill                         !Shut down WebFilesManager object
Hello   PROCEDURE

Window WINDOW,AT(,,139,59),GRAY,DOUBLE         !declare window
   STRING('Hello Web!'),AT(51,14),USE(?Hello)  ! with Hello Web string
   BUTTON('Goodbye Web!'),AT(39,31),USE(?Bye)  ! and Goodbye Web button
       END
WebWindow WebWindowClass                       !Declare WebWindow object

  CODE
  OPEN(window)                                 !Open the window
  WebWindow.Init(WebServer,HtmlManager)        !Initialize WebWindow object by
                                               ! gathering info about window
                                               ! and its controls
  ACCEPT
    IF WebWindow.TakeEvent() THEN BREAK.       !Web event handling:
                                               ! handles all events necessary
                                               ! to respond to Client request
                                               ! e.g. generate new HTML page
    IF EVENT() = EVENT:Accepted                !Usual Windows event handling
   POST(Event:CloseWindow)                     !Close window on ?Bye button
  END
  END
  CLOSE(window)                                !Close the window
  WebWindow.Kill                               !Shut down WebWindow object
  RETURN
```

```
WebControlFactory PROCEDURE(SIGNED Type)       !Instantiate WebControl objects
NewControl    &WebControlClass                 ! requested by WebWindow object
  CODE
  CASE (Type)
  OF CREATE:ClientArea
    NewControl &= NEW WebClientAreaClass
  OF CREATE:String
    NewControl &= NEW WebHtmlStringClass
  OF CREATE:TextButton
    NewControl &= NEW WebHtmlButtonClass
  END
  IF (~NewControl &= NULL)
  NewControl.IsDynamic = TRUE
 END
  RETURN NewControl
```

Hand Coded Project Considerations

The IBC Library requires several components in order to successfully compile and link. Specify the following components with the Project Editor dialog. See *The Project System* in the Online User’s Guide for more information.

ICSTD.CLW

ICSTD.CLW contains a variety of procedures that are shared by several different IBC objects. These procedures are prototyped in ICSTD.INC. These procedures are not methods of a CLASS, and therefore cannot be identified to the linker by the LINK attribute like the IBC methods are. To locate these procedures for the linker, you must add the ICSTD.CLW file to the External source files branch of the project tree. ICSTD.CLW is installed by default to the Clarion LIBSRC\ directory.

DOS Database Driver

The IBC Library objects use the DOS Database Driver to write the HTML code and JSL data requested by Client browsers. You must add the DOS driver to the Database driver libraries branch of the Project tree to resolve IBC references to DOS driver procedures.

ASCII Database Driver

The IBC Library objects use the ASCII Database Driver to process reports. You must add the ASCII driver to the Database driver libraries branch of the Project tree to resolve IBC references to ASCII driver procedures.

C60HTMx.LIB

C60HTMx.LIB contains a variety of compiled objects that are shared by several different IBC objects. These executable objects are prototyped in ICSTD.INC. To locate these executables for the linker, you must add the C60HTMx.LIB file to the **Library, object, and resource files** branch of the project tree.

12 - IBC Library Quick Reference

The Internet Connect Templates rely heavily on the Internet Builder Class (IBC) Library to accomplish the tasks necessary to create a hybrid Web/Windows application. This chapter *briefly* documents the IBC Library methods and properties referenced by the Internet Connect Templates, as well as other IBC Library methods and properties you are likely to use during the course of developing your hybrid Web/Windows application.

For complete documentation of these items and many more, see the *IBC Library Reference*. All the IBC Library methods and properties are fully documented in the *IBC Library Reference*. The *IBC Library Reference* is available in electronic .PDF format on the SoftVelocity web site.

Classes and Their Template Generated Objects

The Internet Connect templates instantiate objects from the IBC Library. The object names are usually similar to the corresponding class names, but they are not exactly the same. As a result, your Web-enabled application's generated code may contain statements similar to these:

```
Broker.Init
MainFrame.TakeEvent
IC:CurFrame.CopyControlsToWindow
WebWindow.OptionBorderWidth = 2
IC:CurControl.Init
IC:CurControl.DisabledAction = DISABLE:Show
WebMenubar.SetBackground(16711680, '')
HtmlPreview.Init(WebServer, HtmlManager, PrintPreviewQueue)
```

The various IBC classes and their template instantiations are listed below so you can more easily identify IBC objects in your application's generated code. The template generated objects are also listed beside the class name in the *Quick Reference* section of this chapter.

Internet Builder Class	Template Generated Object
BrokerClass	Broker
HtmlClass	HtmlManager
JslEventsClass	JavaEvents
TextOutputClass	Target
HttpClass	Broker.Http
WebFilesClass	WebFilesManager, Broker.Files, HtmlManager.Files, Broker.Http.Files, JavaEvents.Files, WebServer.Files, WebWindow.Files, andTarget.Files
WebServerClass	WebServer
WebClientManagerClass	Broker.CurClient
WebFrameClass	MainFrame and IC:CurFrame
WebWindowClass	WebWindow
WebControlClass	IC:CurControl
WebCaptionClass	WebCaption
WebClientAreaClass	WebClientArea
WebMenubarClass	WebMenubar
WebToolbarClass	WebToolbar
WebReportClass	HtmlPreview

Quick Reference

BrokerClass (Broker)

Init (initialize the BrokerClass object)
Kill (shut down the BrokerClass object)
ServerName (server identifier)

WebClientManagerClass (Broker.CurClient)

IP (client IP address)

HtmlClass (HtmlManager)

Init (initialize the HtmlClass object)
Kill (shut down the HtmlClass object)

JslEventsClass (JavaEvents)

Init (initialize the JslEventsClass object)
Kill (shut down the JslEventsClass object)

TextOutputClass (HtmlManager or Target)

Writeln (write one line of text)

HttpClass (Broker.Http)

GetCookie (get cookie from client)
SetCookie (get cookie from client)
SetProcName (set protected area name)
SetProgName (set server name)

WebFilesClass (WebFilesManager or Files)

GetAlias (return HTML alias for file)
Init (initialize the WebFilesClass object)
Kill (shut down the WebFilesClass object)
SelectTarget (set public or secure channel)

WebServerClass (WebServer)

Active (Web mode or Windows mode)
CommandLine (command line parameters)
Connect (open communication channel to Broker)
Init (initialize the WebServerClass object)
JavaLibraryPath (Java Support Library location)
Kill (shut down the WebServerClass object)
PageToReturnTo (return URL)
ProgramName (Server pathname)
Quit (shut down the server program)
SetSendWholePage (force full page refresh)
SetNewPageDisable (suppress outgoing Web pages)
TimeOut (period of inactivity after which to shut down)

WebFrameClass (MainFrame or IC:CurFrame)

CopyControlsToWindow (merge global controls to local window)
FrameWindow (reference to APPLICATION)
TakeEvent (handle browser and ACCEPT loop events)

WebWindowBaseClass (WebWindow)

AllowJava (generate or suppress JavaScript)
BorderWidth (Web page border width)
CloseImage (close button graphic)
CreateCaption (include a titlebar on the Web page)
CreateClose (include a close button on the Web page)
DisabledAction (default HTML for disabled controls)
FormatBorderWidth (HTML table cell border width)
GroupBorderWidth (group box border width)
MenubarType (menu placement)
OptionBorderWidth (option box border width)
SheetBorderWidth (sheet border width)

WebWindowClass (WebWindow)

AuthorizeArea (name of password protected Web page)
HelpDocument (HTML help document)
HelpEnabled (HTML help enabled flag)
HelpRelative (remote or local help document)
IsSecure (public or secure channel)
AddControl (add control information)
CreateHtmlPage (generate HTML for a window)
GetControlInfo (return control reference)
GetToolbarMode (return toolbar entity)
Init (initialize the WebWindowClass object)
Kill (shut down the WebWindowClass object)
MenubarType (menu placement)
SetBackground (set Web page background)
SetFormatOptions (set Web page scale and alignment)
SetHelpDocument (enable single document Web page help)
SetHelpURL (enable multiple document Web page help)
SetPageBackground (set Web page background)
SetPassword (require password)
SetSplash (make this a splash window)
SetTimer (set Web page timer and action)
SuppressControl (omit control from Web page)
TakeEvent (handle browser and ACCEPT loop events)
ValidatePassword (verify password)

WebControlClass (IC:CurControl)

DisabledAction (HTML for disabled control)
CreateHtml (write HTML for control and its attributes)
Feq (control number)
ParentFeq (parent control number)
Init (initialize the WebControlClass object)
Kill (shut down the WebControlClass object)
SetBorderWidth (set BorderWidth)

WebJavaStringClass (IC:CurControl)
SetAutoSpotLink (set live hypertext links)

WebHtmlImageClass (IC:CurControl)
SetDescription (set alternative text for Web image)

WebJavaListClass (IC:CurControl)
ResetFromQueue (record changes to Server LIST queue)
SetAutoSpotLink (set live hypertext links)
SetEventAction (associate browser action with control event)
SetQueue (set the data source queue)

WebCaptionClass (WebCaption)
Alignment (text justification)
SetBackground (set Web page caption background)
SetFont (set Web page caption font)

WebClientAreaClass (WebClientArea)
SetBackground (set Web page client area background)

WebMenubarClass (WebMenuBar)
SetBackground (set Web page menu area background)

WebToolbarClass (WebToolbar)
SetBackground (set Web page toolbar area background)

WebReportClass (HtmlPreview)
Init (initialize the WebReportClass object)
Kill (shut down the WebReportClass object)
Preview (generate HTML to represent the report)

Glossary

All definitions are general terms, except where otherwise indicated. The context for definitions marked (Clarion) pertain specifically to the Clarion language or the Clarion development environment.

applet	A small, single purpose application; applets are not necessarily stand alone executable programs. Small programs written in Java are commonly called applets. In HTML, the <APPLET> tag indicates a Java applet.
Application Broker	(Clarion) An Application Broker is required to run Clarion hybrid Web/Windows applications. The Application Broker launches a hybrid Web/Windows application on the Internet server and refreshes the Clarion Java Support Library (JSL) on the browser. The Application Broker then organizes the message traffic into a remote computing session, routing events produced by the Java Support Library to the hybrid Web/Windows application and routing HTML scripts produced by the application to the browser.
Broker	(Clarion) See Application Broker.
Client	(Clarion) An internet browser that launches a hybrid Web/Windows application with the Application Broker.
Cookie	Information stored on a client machine at the request of a server.
default button	A command button which is activated by default when the user presses the ENTER key.
Disabled	A window, menu, or control visible but prevented from gaining focus.
Encryption	The representation of data in scrambled or encrypted form, such that an unauthorized user may not access the data in an intelligible format.
font	The family name of related type face files. For example, “Times New Roman” is the font name, and “Times New Roman plain,” “Times New Roman Italic,” “Times New Roman Bold,” and “Times New Roman Bold Italic” are the styles, which are stored in separate files.
font style	Character formatting applied to a font face, such as bold, italic, or bold italic.

GIF image	Graphics Interchange File (GIF) format; an image format popularized by CompuServe. Generally acknowledged to offer the best compression ration for 256 color or less images. Attention: should you utilize the word “GIF” anywhere within an application or program, you must add a trademark notice: “GIF (Graphics Interchange Format) is a trademark of CompuServe Information Services.”
global toolbar	A horizontal or vertically arranged group of command buttons, and/or other controls, generally remaining accessible the entire time a program executes.
Hide	Prevent a control or window from displaying on screen; the control exists but is not seen by the end user.
HTML	Hyper-Text Markup Language—the language internet browsers use to format and display Web pages.
HTTP	Hyper-Text Transfer Protocol—the symbols that internet browsers and servers use to transmit and receive HTML.
Hybrid Web/Windows Application	Hybrid Web/Windows Applications look like standard Windows applications when launched under Windows, but work as Internet servers when launched by the Clarion Application Broker. Hybrid Web/Windows applications can then be manipulated from any Java enabled browser such as Microsoft Internet Explorer or Netscape Navigator.
icon	A graphical representation of a physical object in the system, such as a printer. Also, any small image representing an action, concept or program, as when an icon appears on a command button. The normal icon file format carries the .ICO extension; one of its main features is built-in support for transparency. This enables you to display a small picture without obliterating the background.
include file	An external source file read and preprocessed at compile time. In Clarion, the Equates and other files in the LIBSRC subdirectory are the default include files.
Internet Developer’s Kit	(Clarion) The Internet Developer’s Kit is an accessory product that can be used with the Clarion Standard, Professional, or Enterprise Editions to develop new hybrid Web/Windows Applications or to Web-enable existing Clarion applications. A Developer Version of the Application Broker which permits as many as five connections is included with the Internet Developer’s Kit.

Java Support Library	(Clarion) The Java Support Library (JSL) is a small set of Java classes (less than 200k) that implement a wide variey of Windows-like controls in an Internet Browser. The JSL generates events from the internet browser and processes messages from the internet server.
JPG image	A true-color graphics file format featuring 24-bit color storage. It usually provides for adjustable loss compression, which allows for greater compression but loss of some resolution.
JSL data	The protocol and data a hybrid Web/Windows application sends to the internet browser for processing by the Java Support Library (JSL). The hybrid Web/Windows application sends JSL data to the internet browser to accomplish very fast partial Web page updates.
Remote Computing Session	(Clarion) The Clarion Application Broker organizes events produced by the Java Support Library (JSL) and HTML pages produced by hybrid Web/Windows applications into a remote computing session by maintaining the status of the dialog between the browser and server.
Reusable Client	(Clarion) The Java Support Library (JSL) is a small set of Java classes (less than 200K) that generates events from the internet browser and processes messages from the internet server. This thin client is reused by every Clarion hybrid Web/Windows application, thereby minimizing connect time and local browser resource requirements (disk space and RAM).
Server	(Clarion) A hybrid Web/Windows application launched by the Application Broker at the request of an internet browser.
Session Router	(Clarion) The Session Router distributes remote computing sessions to multiple Application Brokers over the Internet, when high popularity or demand requires the deployment of additional Internet servers. The Session Router is available separately.
timer	A Windows resource which can automatically send a message to an application at pre-defined intervals.
Ultra-thin Reusable Client	(Clarion) The Java Support Library (JSL) is a small set of Java classes (less than 200K) that generates events from the internet browser and processes messages from the internet server. This thin client is reused by every Clarion hybrid Web/Windows application, thereby minimizing connect time and local browser resource requirements (disk space and RAM).

Index: