

CLARION 5

**Internet
Connect**

COPYRIGHT 1997, 1998 by TopSpeed Corporation
All rights reserved.

This publication is protected by copyright and all rights are reserved by TopSpeed Corporation. It may not, in whole or part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from TopSpeed Corporation.

This publication supports Clarion 5 Internet Connect. It is possible that it may contain technical or typographical errors. TopSpeed Corporation provides this publication "as is," without warranty of any kind, either expressed or implied.

TopSpeed Corporation
150 East Sample Road
Pompano Beach, Florida 33064
(954) 785-4555

Trademark Acknowledgements:

TopSpeed® is a registered trademark of TopSpeed Corporation.

Clarion™ is a trademark of TopSpeed Corporation.

Microsoft® Windows®, Windows 95®, Windows 98®, and Windows NT® are registered trademarks of Microsoft Corporation.

All other products and company names are trademarks of their respective owners.

TABLE OF CONTENTS

INTRODUCTION	7
Welcome!	7
What is Internet Connect	8
Internet Connect and the Clarion Development Environment	8
What You'll Find in this Book	10
Where to Find More Information	11
Documentation Conventions	12
Typeface Conventions	12
Keyboard Conventions	12
Product Information	13
Registering This Product	13
Technical Support	13
SETUP	15
System Requirements	15
Development System	15
Server System	15
Client System.....	16
The Setup Program	17
Starting Setup	17
Setup Options	17
Registering the Internet Builder Class (IBC) Templates	18
THE APPLICATION BROKER	19
Running the Application Broker	20
Using the Single-connection Executable Version	20
Connecting to your Applications	21
The ISAPI version of the Application Broker	23
How the Application Broker ISAPI DLLs work	23
Installation and Setup for the Clarion Application Broker ISAPI DLLs	24
The ISAPI Broker's Remote Setup Utility	27
Files deployed by the Clarion Application Broker installation	27

Directories	28
Application Broker Setup Options	29
Remote Access to the Application Broker	31
Development & Deployment CheckList	33
Testing Locally	35
Testing your TCP/IP Connection	36
TUTORIAL—MAKING A WEB APPLICATION	37
Web Application Wizard	38
Creating a hybrid Web/Windows Application	38
Deploying the Application	41
Faster is Better—Optimizing your Application	44
Looks are Important—Adding Graphics	47
TUTORIAL—WEB-ENABLING AN EXISTING APPLICATION	49
Using the Global Internet Application Extension Template	50
Porting an Application to the Web	50
TUTORIAL—ADVANCED WEB PROGRAMMING TECHNIQUES	55
Using Cookies	56
Embedding HTML	61
Covering the Download with a Splash Window	65
Using Partial Refresh to Update Controls	68
Restricting Access to a Procedure	70
Password Protection	70
Restricting Edit-In-Place	73
THE INTERNET BUILDER CLASS TEMPLATES	75
The Global Internet Application Extension Template	75
Page Settings	76
Window Settings	76
Help	77
Control	78

MDI	79
Advanced	80
Classes	81
Global Window Component Options	82
Caption	82
Menu	83
ToolBar	84
Client Area	85
Class Overrides	85
Internet Procedure Extension Template	86
Page Settings	86
Window Settings	87
Help	87
Controls	88
MDI	90
Advanced	90
Individual Overrides for a Control	93
Display	93
HTML	94
Events	95
Classes	96
Procedure Window Component Options	97
Caption	97
Menu	98
Toolbar	98
Client Area	99
Frame Procedure MDI Options	101
Application Menu	101
Application Toolbar	101
Code Templates	103
Dynamic HTML Code Template	103
Static HTML Code Template	103
GetCookie Code Template	104
SetCookie Code Template	104
Cookies (Persistent Client Data)	104
AddServerProperty Code Template	105
GetServerProperty Code Template	105

WEB APPLICATION DESIGN

CONSIDERATIONS	107
Bandwidth Usage Considerations	107
Use Partial Refresh whenever possible	107
Be frugal with controls	108
Use graphics sparingly	108
Covering the Download with a Splash Window	109
Cosmetic Design Considerations	112
Using Groups	112
Using Images	113
User Interface Design Considerations	114
MDI window access	114
Restricting Edit-In-Place	114
Unsupported Windows Standard Dialogs	115
Using Command Line Parameters	116
Changing the Class for an individual control	116
API calls	117
Security Considerations	118
Using Passwords	118
Using a Secure Socket Layer (SSL)	119
Using Embedded HTML	120
Using references to files in embedded HTML code	121
Implementing Help in your Web Application	123
Using a Base Document with Mid-Page anchors	123
Using individual help Documents	123
Windows Controls and their HTML Equivalents	125
Hand Coded Applications	128
About This Section	128
HelloWeb Example Program	128
Hand Coded Project Considerations	130
IBC LIBRARY QUICK REFERENCE	133
Classes and Their Template Generated Objects	134
Quick Reference	135
GLOSSARY	139
INDEX	143

INTRODUCTION



This chapter provides:

- ◆ An introduction to Clarion Internet Connect.
- ◆ An overview of what you'll find in the *Internet Connect User's Guide*.
- ◆ Typeface and other document conventions.
- ◆ A reminder about product registration.
- ◆ A summary of our technical support programs.

Welcome!

Thank you for purchasing Clarion Internet Connect—TopSpeed's one-step Web solution!

This book contains *three* tutorials, on three different levels:

- ◆ A short **Making a Web Application** tutorial, which introduces the quickest way to create a database application for the Internet/Intranet.
- ◆ A **Web-enabling an Existing Application** tutorial, which shows you how to port your existing Clarion applications to the Web.
- ◆ An **Advanced Web Programming Techniques** tutorial, which shows some methods of extending the built-in functionality using code templates and embedded HTML.

This book assumes you have completed the tutorials in the Clarion *Getting Started* and *Learning Clarion* manuals. If you have not yet done so, we urge you to do them before starting the tutorials in this book.

It is also useful to be familiar with the way Web browsers work. Some basic HTML knowledge is also useful, but not required.

What is Internet Connect

Clarion and Clarion Internet Connect work together 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 Java-enabled browser.

You can run a web-enabled application on any platform for which a Java-enabled browser is available.

Internet Connect and the Clarion Development Environment

The development environment works on several levels and supports every level of user/developer.

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.

Anybody can do this. It just starts with picking a data file from a list.

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.

Power users, who may not normally write programs, can easily do this.

It's a complete programming language

At its most basic level, you can completely hand-code an application. The Clarion language is a fourth generation language. It has a high level of abstraction, so that it's very "readable," and a compact database grammar, so that you can easily handle a record or sets of records. You don't *have to* know Clarion to create an application ... but if you do know how it works, it helps you understand what your applications are doing, and that helps you make better applications.

The Internet Builder Class Library is open and available to web-enable a hand-coded project.

Professional developers will really appreciate the Clarion language. It was designed from the ground up for business programmers. Yet for its relatively low learning curve (as a 4GL, vs. lower level languages) you'll get blazing performance. The TopSpeed compiler turns all of your code—whether you wrote it or the Application Generator wrote it—into highly optimized machine code.

No matter which level you intend to work at, you're going to work a lot smarter if you read this book all the way through to the end.

What You'll Find in this Book

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

Introduction

Chapter One: the chapter you're reading now. This is an introduction to web programming using Clarion.

Setup

Chapter Two: step by step instructions tell you how to install Clarion Internet Connect to your Clarion development environment.

The Application Broker

Chapter Three: documents the Clarion Application Broker, which is the key to running web-enabled Clarion applications across the Web.

Application Wizard Tutorial

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

This chapter will show you how to use the Application Wizard and set some of the Internet Options to create a simple application that will run either locally or over the Web.

You'll accomplish all this *without writing a single line of code*.

Web-enabling an Existing Application

Chapter Five: using the IBC templates to port Clarion applications to the Web.

Advanced Web Programming Techniques

Chapter Six : 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 Seven: A reference to the IBC Template interface.

Application Design Considerations

Chapter Eight: Tips and techniques on web-based application design.

Internet Builder Class Library- A Quick Reference

Chapter Nine: 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

Where to Find More Information

The first place to look for more information is the **How do I ... ?** section in the online Help. These topics answer many of the common questions that newcomers to Clarion have. Click on the **How do I ... ?** link on the Help Contents page to get to this section.

The *Application Broker Reference* (Chapter 3 of this book) is the guide to installing, configuring, and using the Clarion Application Broker.

The *Internet Builder Class Library Reference* (on CD in .PDF format) is the complete guide to the classes used by the IBC templates. It provides descriptions of all the classes, methods, and properties with examples for each.

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. TopSpeed Corporation 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

<i>Italics</i>	Indicates what to type at the keyboard, such as <i>Enter This</i> .
SMALL CAPS	Indicates keystrokes to enter at the keyboard, such as ENTER or ESCAPE, or to CLICK the mouse.
Boldface	Indicates commands or options from a pulldown menu or text in a dialog window. Note: this style also utilizes a different typeface to match the helvetica bold face which Windows uses as the system font.
LETTER GOTHIC	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 Clarion Internet Connect, 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 TopSpeed's Technical Support services and you automatically receive new product announcements and update alerts.

Technical Support

There are several venues for technical support:

CompuServe

You can receive unlimited free technical support for Clarion on CompuServe Information Service. Once connected to CompuServe, type GO TOPSPEED. TopSpeed employees, as well as TopSpeed Certified Support Partners (known as Team TopSpeed), will answer your questions in a timely manner. Additionally you can get advice and answers from other Clarion users. We strongly recommend that our customers take advantage of this service.

Usenet Newsgroup--comp.lang.clarion

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

You can use any newsreader (e.g., Outlook Express, Netscape Collabra, or FreeAgent, etc.) or you can access Usenet newsgroups through services such as DejaNews (<http://www.dejanews.com>). DejaNews archives Usenet newsgroups and offers search capabilities.

TopSpeed's product newsgroups

TopSpeed's internal newsserver offers newsgroups for all TopSpeed products.

News server: TSNews.Clarion.Com

Newsgroup: TopSpeed.Products.IC

TopSpeed's Web Site:

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

<http://www.topspeed.com>

Click on the **Downloads** link to see the latest patches for all of TopSpeed products.

Click on the **What's New** link to read news and announcements.

Paid Technical Support

Paid telephone technical support is also available from TopSpeed Corporation. Call TopSpeed Corporation customer service at (800) 354-5444 or (954) 785-4555 for more information.

SETUP

2

System Requirements

Development System

You can run the Clarion development environment on any system that meets the minimum system requirements for Windows 95™, Windows 98™ or Windows NT. Web-enabled applications must be compiled in 32-bit, so you must be running Windows 95, Windows 98 or Windows NT.

- ◆ Windows 95/98, 16 (or more) Megabytes of RAM recommended.
- ◆ Windows NT, 32 (or more) Megabytes of RAM recommended.
- ◆ Minimum of 7 to 13 Megabytes free hard disk space, depending on the Setup options you select.

Server System

You can run the Application Broker and Web-enabled applications on Windows 95/98; however, we *strongly* recommend using Windows NT to host the deployment.

- ◆ Windows 95/98, 32 Megabytes of RAM recommended, static connection to Internet/Intranet.
- ◆ Windows NT, 64 Megabytes of RAM recommended, static connection to Internet/Intranet.

Performance depends on the speed of your server's connection to the Internet and the traffic you expect your application to handle. Applications can be delivered over a 28.8 kb modem connection, but we recommend ISDN or higher.

Client System

Clients can run under any platform for which a Java-enabled browser is available. The applications that you develop with Clarion Internet Connect will execute comfortably on computers that meet only the minimum requirements for these browsers. Performance is affected by the speed of the connection to the Internet, but most applications will perform well over a 28.8 kb modem connection.

Recommended Browsers:

Although Web-enabled applications should work under any Java-enabled browser, the following browsers have shown the best results.

Windows 95/NT

Microsoft Internet Explorer 3.0x or later (version 3.02 introduced Java Virtual Machine enhancements which improve memory allocation).

Netscape Navigator 3.0 or later.

Netscape Communicator 4.0 or later. (version 4.04 introduced fixes that improve performance).

Windows 3.1x

Microsoft Internet Explorer 3.02x or later.

Netscape Navigator 3.0 or later.

Netscape Communicator 4.0 or later.

UNIX

There are many flavors of UNIX and some of these have browsers written specifically for them. There are too many to list here. See Netscape's Home page at <http://www.netscape.com> for more information.

Netscape Navigator 3.01 or later.

Netscape Communicator 4.03 or later.

Apple/Macintosh

Microsoft Internet Explorer 3.0x or later.

Netscape Navigator 3.0 or later.

Netscape Communicator 4.0 or later.

OS/2

Netscape Communicator 4.0 or later.

Note: The installation installs the JSL files in compressed format (Clarion.CAB and Clarion.ZIP). The installation also installs the .CLASS files to support 16-bit browsers and browsers for non-Windows platforms.

The Setup Program

The Setup program decompresses and copies the Clarion files to your hard drive. For all the target operating systems, it provides you with options for installing the various components, such as the example files.

Starting Setup

To start the Clarion Setup program in Windows 95/98 or NT:

1. Insert the installation CD into your CD-ROM drive.
2. From the Start menu, choose **Settings ► Control Panel**.
3. Choose **Add/Remove Programs** then press the **Install** button.

The Windows 95/98/NT Wizard directs you through the installation process.

Setup Options

After starting Setup, you'll see a set of wizard screens displaying a number of options.

1. Choose the Setup options offered by the wizard screens (such as the target drive and directory), pressing the **Next** button to move through the option screens.

Setup will install the components of the Clarion Internet Developer's Kit to the appropriate subdirectories below the target directory you specify. The Clarion Setup program installs *all* its files to the target directory, and subdirectories beneath it. It installs *no* files to any other directory, with one exception. The ISAPI Version of the Application Broker requires that one file (CWISAPI.DLL) be installed in the \WinNT\System32 directory on a Windows NT machine or the \Windows\System directory on a Windows 95/98 machine.

During the installation, progress bars display as Setup copies the files.

2. Press the **OK** button when Setup is done.

Registering the Internet Builder Class (IBC) Templates

To use the Internet Developer's Kit's IBC Templates, register the *ICONNECT.TPL* file in the Clarion Template Registry. The file is installed in the `..\TEMPLATE` subdirectory.

To register the IBC Templates:

1. Choose **Setup ► Template Registry** from the Clarion menu.
2. In the Template Registry dialog, press the **Register** button.
3. In the Template File dialog, select the *ICONNECT.TPL* file in the `..\TEMPLATE` subdirectory, then press **OK**.
4. In the Template Registry dialog, press the **Close** button.

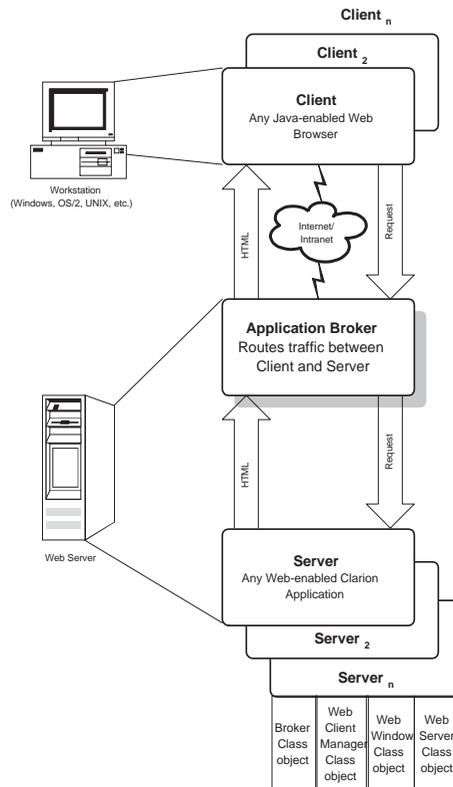
THE APPLICATION BROKER

3

This chapter covers the the Application Broker—your connection to Internet database applications made with Internet Connect.

The Internet Developer's Kit is made up of two parts: The Internet Builder Class (IBC) Templates/Library and the Application Broker. The templates Web-enable your Clarion application and the Application Broker allows access to that application over the Web.

Using the included Clarion Application Broker—no other Web server software required—allows end-users to manipulate Web-enabled applications running on a Web server using a browser.



Running the Application Broker

There are two versions of the Application Broker—an executable version (.EXE) and an ISAPI DLL version (.DLL). The executable version of the Application Broker must be executed to start it. The ISAPI version is called on demand by the ISAPI-compliant Web server. See *The ISAPI version of the Application Broker* for more information.

7. Start the Application Broker by DOUBLE-CLICKING on *CWBrokr1.exe* (or *CWBroker.exe* if you have the multiuser version of the Application Broker).

When you run the Application Broker on the server machine, it defaults to port 80. If you are running any other Web server which is using port 80, you can specify another port (e.g., 8080) for the Application Broker in its setup options or by creating a shortcut with the port as a command line parameter.

Note: If you specify the port number on the command line, you cannot change it in the Broker's setup options. If you start the Application Broker, allow it to warn you that it cannot use the default port, then change the port setting, it saves that setting and you will not need to specify the port the next time it executes.

Using the Single-connection Executable Version

The Internet Developer's Kit contains a single-connection Application Broker (*CWBrokr1.EXE*). It also contains a single-connection ISAPI version of the Application Broker. This version is designed for testing.

The single-connection Application Broker functions exactly like the Application Broker, except that it only allows only access by a *single IP address* at a time. That means one client can run more than one copy of an executable or run multiple executables by opening another browser window. To allow access to another user, all applications started by the first client must be closed.

A client must exit the application for the application to close; simply closing the browser does not close an application. If the client does close the browser, the application won't close until a timeout occurs. The templates set an application's timeout to 10 minutes by default. You can modify this timeout interval in the *Global Internet Application Extension Template*.

Testing Concurrent Access

You can easily test any concurrency issues (i.e., issues in your application concerning more than one user) with the single-connection Application

Broker by starting two instances of your browser and running a copy of the application in each.

To start a new instance of your browser:

In Internet Explorer 3.0x:

1. Choose **File** ► **New Window**.

In Internet Explorer 4.0x:

1. Choose **File** ► **New** ► **Window**.

In Netscape Navigator 3.0x:

1. Choose **File** ► **New Web Browser**.

In Netscape Navigator 4.0x:

1. Choose **File** ► **New** ► **Navigator Window**.

Connecting to your Applications

To run a Web-enabled application from a browser, provide a Uniform Resource Location (URL) in the following format:

```
http://nnn.nnn.nnn.nnn/appname.exe.0
```

where *appname.exe* is your application's executable file name (make sure to add a dot zero (.0) after the executable name) and *nnn.nnn.nnn.nnn* is your IP address, or

```
http://domain.ext/appname.exe.0
```

Where *domain.ext* is your domain name, if your IP address has a name registered with a Domain Name Server (DNS).

If you are running the broker on a port other than port 80 (the default HTTP port), include the port number in the URL.

Examples:

```
http://nnn.nnn.nnn.nnn:8080/appname.exe.0  
http://domain.ext:8080/appname.exe.0
```

Using Hyperlinks to start an application

You can also create hyperlinks in standard HTML Web page to execute Web-enabled applications.

For example:

```
<A HREF="http://nnn.nnn.nnn.nnn/appname.exe.0">Click to start app</a>
```

or

```
<A HREF="http://nnn.nnn.nnn.nnn:8080/appname.exe.0">Click to start app</a>
```

Firewalls and alternate ports

Firewalls and proxy servers can be configured to restrict access to ports other than the default HTTP port (80). To facilitate access for users behind firewalls or proxy servers, you have the following options:

1. Use the ISAPI Version of the Application Broker and run your Web server on port 80.
2. Run the executable Application Broker on port 80.
3. Have the network administrator remove the restriction to port 8080 (or the port you are using).

The ISAPI version of the Application Broker

Internet Connect also has an Application Broker which uses Internet Server Application Programming Interface (ISAPI) Server Extensions. You can use this version with Microsoft's Internet Information Server (IIS), Personal Web Server (PWS), or any other ISAPI compliant Web server.

This version of the Clarion Application Broker is designed to implement security using Secure Socket Layer (SSL) encryption. This technology uses advanced cryptography techniques to encrypt data sent and received over the Internet. This makes it almost impossible for others to decipher intercepted information. When you see the gold "lock" icon in the status bar of your Web browser, you know that you are on a secure layer and the data sent and received is private and secure.

Note: Personal Web Server does not provide SSL encryption, so no security is provided when you use the ISAPI Application Broker with Personal Web Server. This just allows you to design and run your applications on a Windows 95/98 machine and deploy it later on a Windows NT machine running IIS. Personal Web Server is on the Windows 98 installation CD or it can be downloaded from www.microsoft.com.

How the Application Broker ISAPI DLLs work

The Clarion Application Broker is implemented as an ISAPI Server extension Dynamic Link Library (DLL). A request from a client (browser) is made to the Web server specifying the base DLL and the Web-enabled application to run. This loads the Application Broker into memory and it maintains the connection between the client and the server application the same way the executable version does.

When your Web-enabled application makes a secure request (specified in the Internet Options for a procedure), the Application Broker accepts the request over a secure sockets layer and remains in secure-mode while that procedure is active and until a procedure without security enabled is called.

You should use encryption sparingly. Secure Sockets Layer (SSL) slows performance between HTTP servers and browsers. For this reason, Internet Connect allows you to switch between a normal and secure mode at any time in your application. Using encryption only where you need security *dramatically* improves performance.

Installation and Setup for the Clarion Application Broker ISAPI DLLs

1. Install Microsoft Internet Information Server (IIS) or Personal Web Server (PWS) first.
2. Optionally, obtain a Secure Sockets Layer (PCT/SSL) Digital Certificate and install it. You can obtain a certificate from <http://www.verisign.com>. They also offer a free trial certificate for testing purposes. See the help file for IIS's Key Manager for details on installing a certificate.

Note: You can use the ISAPI DLLs without implementing SSL. If you do not want to use SSL, do not enable SSL when installing the Application Broker and skip Step 2. You can change this setting later using the Application Broker's administrator utility (ICCONFIG.EXE).

3. Install the ISAPI DLL version of the Clarion Application Broker. The installation prompts you for the following information:

Install Drive The drive to which the broker is installed.

Root directory for the install

The root directory for the install. This should not be the root of your Web site.

The Application Broker Password

This is the password needed to access the broker setup. You must provide a password.

Server IP address or Domain Name

The Domain Name of the server machine or its IP address.

Enable SSL

If you intend to use SSL and have a digital certificate, check this box. If you do not want to use SSL or don not have digital certificate, clear this box.

Note: PWS does not support SSL or digital certificates.

The installation creates these directories below the specified Root Installation directory:

- \scripts
- \sec_scr
- \secure
- \public
- \exec
- \admin

The installation also creates a text file (CWISSET.TXT) listing the directories created and a list of the the settings you must make in your Web Server.

4. Using the IIS Internet Service Manager utility or PWS Administrator (in Control Panel), set the access rights for the directories as listed in the CWISSET.TXT file.

For example, using C: as the Install drive and CWICWeb as the Install Root, you **must** set these rights under IIS:

<u>Directory</u>	<u>Alias</u>	<u>Rights</u>
C:\CWICWeb\public	/cwpub	Read
C:\CWICWeb\secure	/cwsec	Read & SSL
C:\CWICWeb\scripts	/cws	Execute
C:\CWICWeb\sec_scr	/cwss	Execute & SSL

Note: SSL settings are only appropriate when using a digital certificate. You can run the ISAPI version or the Application Broker without a certificate but you cannot use SSL.

The installation also creates these directories, but no setting need be made in IIS for them:

<u>Directory</u>		
C:\CWICWeb\exec	none*	none*
C:\CWICWeb\admin	none*	none*

5. Using NT's User Manager utility, grant WRITE and DELETE (Change) rights to the Internet Guest account for the \Public and \Secure directories.

Note: Using Personal Web Server, no User access rights need to be set.

6. Deploy your Web-enabled applications to the C:\CWICWeb\Exec directory (or a directory beneath it). You must also deploy any support DLLs (file drivers, runtime libraries, etc.) into the same directory as the executable.

Note: The 32-bit Clarion Runtime Library (C5RUNx.DLL, the 32-bit DOS file driver (C5DOSx.dll) and the 32-bit ASCII file driver (C5ASCx.DLL) are also required even if you do not use them in your .APP (unless compiled as a Local Link executable).

7. Deploy any Icon files needed for your application to the C:\CWICWeb\Public directory and the C:\CWICWeb\Secure directory. Icons displayed in LISTS or on BUTTONs must be deployed to these directories.

The \Secure directory is used when a procedure is set to use SSL. See the *Internet Procedure Extension Template* section in the *Internet Extension Templates* chapter.

Optionally, deploy any .GIF or .JPG files your applications use to these directories, too. A Web-enabled application will automatically extract image files from IMAGE controls at runtime, but if they are deployed in advance, it can skip that step. See *Using Images* in the *Application Design Considerations* chapter.

- ℘ Optionally, create your Web page to call your applications using this convention for the hyperlinks:

```
<A HREF="http://domainname.com/cws/cwlaunch.dll/AppName.exe.0">Click to start app</a>
```

The ISAPI Broker's Remote Setup Utility

To run the Application Broker's setup utility.

- From a browser, call this URL:

```
http://nnn.nnn.nnn/cws/cwlaunch.dll/AppBroker/
http://domainname.com/cws/cwlaunch.dll/AppBroker/
```

Note: This **MUST** contain the trailing backslash.

The browser's authentication dialog appears.

- Type in *CWIC* for the Username and type in your password (see *Application Broker Setup Options*), then press the **OK** button.

The Application Broker's remote access page appears. This page contains hyperlinks to control the broker remotely.

- Click on the **Setup** hyperlink.

This calls the remote setup page. See *Remote Access to the Application Broker* for details. This allows you to modify your Application Broker settings. Check these now to make sure the entries are correct.

Files deployed by the Clarion Application Broker installation

<u>File Name</u>	<u>Directory</u>
CWLAUNCH.DLL	C:\CWICWeb\scripts
CWSECURE.DLL	C:\CWICWeb\sec_scr
CWISAPI.DLL	\WinNT\System32 or \Windows\System
CLARION.ZIP	C:\CWICWeb\public & C:\CWICWeb\Secure
CLARION.CAB	C:\CWICWeb\public& C:\CWICWeb\Secure
*.class	C:\CWICWeb\public & C:\CWICWeb\Secure

The \Secure directory is used when a procedure is set to use SSL by checking the **Transfer over a secure connection** box in the Advanced Options. See the *Internet Procedure Extension Template* section in the *Internet Extension Templates* chapter.

Note: The installation installs the JSL files in compressed format (Clarion.CAB and Clarion.ZIP). The installation also installs the .CLASS files to support 16-bit browsers and browsers for non-Windows platforms.

Directories

EXE Version

When you run the executable version of the Application Broker, the directory from which it is run is the virtual root directory for executables and the \Public directory below that directory is the virtual root for any downloadable files. For example, a link to an HTML document in the \Public directory would be addressed as:

```
http://nnn.nnn.nnn.nnn/index.htm
```

while an executable in the same directory as the Application Broker is referenced using a similar URL:

```
http://nnn.nnn.nnn.nnn/appname.exe.0
```

Even though these files are in different directories, the Application Broker handles the routing and no relative directory information is required.

If you want to keep applications in different directories, you call the application using its *relative* path. For example, with the executable version of the broker running in C:\CWICWeb and an application named myapp.exe is deployed to C:\CWICWeb\Exec\App1. You would call it using:

```
http://mydomain.com/exec/app1/appname.exe.0
```

ISAPI Version

With the ISAPI version of the Application broker, the directories and their aliases are specified in the Application Broker's setup options and in the Web server's directory setup.

If you want to keep applications in different directories, you call the application using its *relative* path. Using the ISAPI version of the broker with C:\CWICWeb\Exec specified as the executable directory and an application named myapp.exe is deployed to C:\CWICWeb\Exec\App1. You would call it using:

```
http://mydomain.com/cws/cwlaunch.dll/app1/appname.exe.0
```

Application Broker Setup Options

From the Application Broker's Setup menu option, you can specify the following options:

Default Home Page

The document which is delivered by default if no specific URL is specified. The default is *index.htm*.

Public Directory

The directory where common deliverable files are deployed (HTML files, images, etc.). This is also the directory under which temporary directories containing the HTML files representing an application are created. The default is */Public*.

Port Number

The HTTP port to which the application broker is bound. If specified on the broker's command line, this is disabled and cannot be modified.

When a port other than 80 (the default HTTP port) is used, clients (or your hyperlinks) must specify the port in the URL. For example if the broker is attached to port 8080, you would specify:

```
http://nnn.nnn.nnn.nnn:8080/appname.exe.0
```

Use Log File

Checking this box enables server logging. This generates a logfile entry for each request made by clients accessing the Application Broker. The logfile contains the requester's IP address, the date and time of the request, and the request made. Keep in mind that the logfile will continue to grow as entries are appended to it.

Log File Name

Specifies the name of the logfile.

Remote password

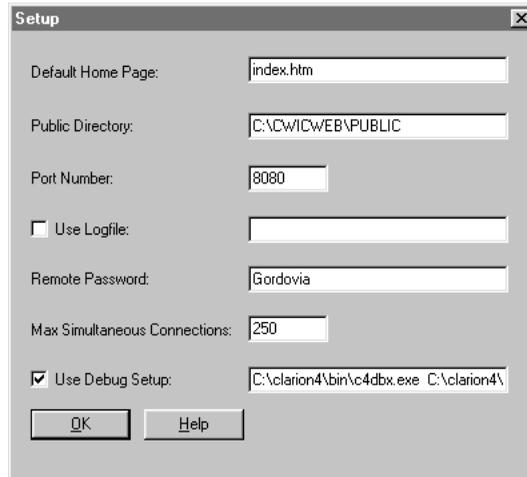
Specify a password to enable remote access to the Application Broker. See *Remote Access to the Application Broker* for more information.

Max Simultaneous Connections

This specifies the maximum number of connections allowed. A notice is returned to users trying to access the broker when an attempt to exceed this number is made.

Use Debug Setup This specifies to run applications in the debugger. You must have access to the Server machine to run the debugger. Specify the location of the debugger and redirection file in the entry field provided. For example:

C:\clarion5\bin\c5dbx.exe C:\clarion5\bin\clarion5.red



Setup

Default Home Page: index.htm

Public Directory: C:\CWIC\WEB\PUBLIC

Port Number: 8080

Use Logfile:

Remote Password: Gordovia

Max Simultaneous Connections: 250

Use Debug Setup: C:\clarion4\bin\c4dbx.exe C:\clarion4\

OK Help

Remote Access to the Application Broker

If you provide a Remote password in the Application Broker Setup, you can access your running broker from a remote site using a browser. This allows you to modify settings, suspend and enable access to an application, and see the status of the last request.

To access your broker from a browser:

1. Type the following URL in your browser's Address or Location entry control, then press ENTER.

```
HTTP://domainname.ext/appbroker/
```

or

```
HTTP://nnn.nnn.nnn.nnn/appbroker/
```

where *domainname.ext* is your server Domain Name and extension or *nnn.nnn.nnn.nnn* is your server IP address. You must include the trailing slash (/).

A password dialog appears.

2. Type CWIC in the Username field and the password you specified during the installation in the broker's Remote Password field.

Note: You cannot use remote access to the Application Broker unless you have specified a Password in the broker's setup.

A Web page appears with hyperlinks for the following functions:

Setup	Provides access to modify the setup options. You can modify any of the setup options except the Port Number and the Remote Password. Those settings can only be changed on the server.
Suspend	Allows you to suspend access to any Web-enabled application deployed on the Application Broker site. If an application is running, the session is closed and the application is closed. Users who were running the program receive a message informing them that the application has been suspended for maintenance. This allows you to deploy updated versions of applications or data files.

To suspend access, type in the full name of the executable file (including the extension), then press the OK button. If the application to suspend is deployed to a subdirectory, you need not specify that directory here. A Web page is returned listing the applications which are currently suspended.

- Enable** Allows you to re-enable access to any Web-enabled application on the Application broker site which has been suspended.
- To re-enable access, type in the full name of the executable file (including the extension), then press the OK button. A Web page is returned listing the applications which are still suspended.
- Status** This displays the current status of applications running on your broker site including the time of access and the IP address of the clients accessing applications. The ISAPI broker also provides a Terminate button for each instance of an application.

Development & Deployment Checklist

1. Create an application (or open an existing application).
2. Web-enable the Application by adding the Global Internet Application Extension Template. See *Using the Global Internet Application Extension Template*.
3. Compile the application in 32-bit mode (either local or standalone).
4. Install and configure the Application Broker (*CWBroker.exe* or *CWBrokr1.exe* or the ISAPI DLL version). Remember that single user Application Broker is for testing only.
5. If you are running the executable version, start the Application Broker on the server (this cannot be started remotely). If you are running the ISAPI version of the Broker, make sure your WWW Service is running.
6. Deploy the application and any other files it needs (DLLs, data files, image files, etc.) to the directory from which the Application Broker will run (or a subdirectory).

Note: The 32-bit Clarion Runtime Library (C5RUNx.DLL, the 32-bit DOS file driver (C5DOSx.dll) and the 32-bit ASCII file driver (C5ASCx.DLL) are also required even if you do not use them in your .APP (unless compiled as a Local Link executable).

Executables and their support files must be in the same directory as the application, in a directory in the PATH (Windows 95/98) or in the Internet Guest Account's system path (Windows NT). In other words, supporting DLLs must be visible when the server application runs. Data files should be in the same directory as the executable, or if your application is created to use data files in a different directory or drive, this location must be visible to the application.

7. The install creates a subdirectory below the broker directory named \Public and a directory named \Secure. These directories are used to deliver files (such as the Java Classes from the Java Support Library, graphics or other HTML files) by the Application Broker.

The \Secure directory is used when a procedure is set to use SSL by checking the **Transfer over a secure connection** box in the Advanced Options. See the *Internet Procedure Extension Template* section in the *Internet Extension Templates* chapter.

The Application Broker will not deliver any files from its executable directory. This prevents downloading data files or executables from your site. Your application creates HTML files at runtime and automatically places them in a temporary subdirectory below either the \Public or \Secure directory.

Note: The installation installs the JSL files in compressed format (Clarion.CAB and Clarion.ZIP) and in uncompressed format to support 16-bit browsers and browsers on non-Windows platforms.

At runtime, a temporary directory is created for each client connection. These directories are automatically deleted when the connection terminates. Graphics in IMAGE controls are automatically extracted to this directory when they are not found in the /Public directory. Icons displayed in LISTS or on BUTTONs are not automatically extracted and must be deployed to the /Public directory and the /Secure directory.

- ℘ Run the Application Broker on the server machine. It defaults to port 80. If you are running any other Web server which is using port 80, you can specify another port (e.g., 8080) for the Application Broker by creating a shortcut with the port as a command line parameter. You can also modify the port to which it is bound by using the Application Broker's Setup option.

Note: If you specify the port number on the command line, you cannot change it in the Broker's setup options. If you start the Application Broker, allow it to warn you that it cannot use the default port, then change the port setting, it saves that setting and you will not need to specify the port the next time it executes.

- ℘ Provide users who want to access the application a URL in the following format:

`http://nnn.nnn.nnn.nnn/appname.exe.0`

or

`http://domain.ext/appname.exe.0`

appname.exe is your application's executable file name. Make sure the user adds a dot zero (.0) after the executable name.

If you are running the broker on a port other than 80 (the default HTTP port), users must include the port number in the URL. For example,

`http://nnn.nnn.nnn.nnn:8080/appname.exe.0`

- If your application has any command line parameters, add a question mark and the command line parameter.

For example:

`http://nnn.nnn.nnn.nnn/appname.exe.0?myargument`

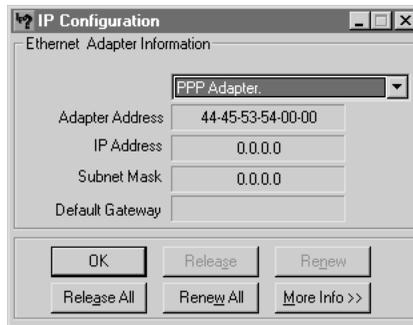
See Also: *Application Design Considerations: Using Command Line Parameters, Testing Locally, Testing your TCP/IP Connection*

Testing Locally

Although for true deployment of a Web-enabled application you will use a persistent IP address, you can use an IP address that is dynamically assigned when you connect using Dial-up Networking. This is useful to provide a one-time demo over the Internet. To find your dynamic IP address, you can run an applet included in your operating system.

To test locally (with the broker and browser on the same machine), you can use the localhost loopback IP address: 127.0.0.1.

In Windows 95, run *Winipcfg.exe* (in the Windows directory). It will show you your address as shown below. If you have TCP/IP bound to a network adapter and a modem, use the drop down to select the modem.



In Windows NT, use the Dial Up Networking Manager to see your IP address. After dialing up and connecting to your provider, click on the icon in the System Tray.

Testing your TCP/IP Connection

There are several methods to test your TCP/IP Connection.

1. From a DOS Prompt, type

```
PING 127.0.0.1
```

This tests the TCP/IP connection on your local machine and returns the time it took to respond.

2. Start your Web browser, type the following URL to the server machine and target test page:

```
http://server.domain.com/index.htm
```

where server.domain.com is the IP address of your server.

If you assigned a different port to your Web server (other than the default port 80), you must also include the port number in the URL, as shown here:

```
http://server.domain.com:XX/index.htm
```

where XX is the port number assigned to your Web server.

If you don't have a TCP/IP connection or you want to test the connection from your local machine, you can use the following URL convention:

```
http://localhost/index.htm
```

or

```
http://127.0.0.1/index.htm
```

This is the internal local loopback address.

TUTORIAL—MAKING A WEB APPLICATION

4

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:\Clarion5 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* manuals 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. Select C:\Clarion5\Examples\WebTutor from the Directories list.
3. Type *WebOrder* in the **File Name** field.

4. Make sure the **Use Quick Start Wizard** box is not checked, then press the **Save** button.

This opens the **Application Properties** dialog.

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

This opens the **Select Dictionary** dialog.

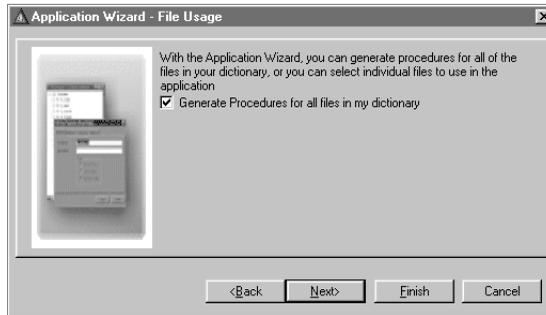
6. Highlight the *WebOrder.dct* (in the C:\Clarion5\Examples\WebTutor\ directory) file then press the **Open** button.

Run the Application Wizard

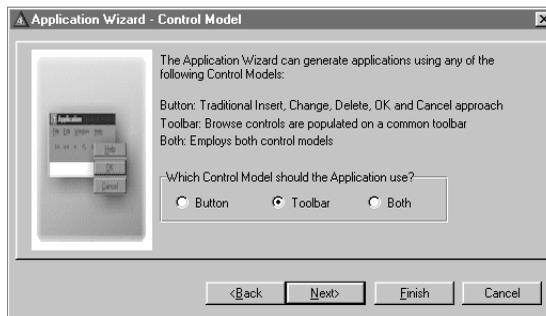
7. Check the **Application Wizard** box, then press the **OK** button.



2. Press the **Next** button.



3. Press the **Next** button.



4. Press the **Next** button.



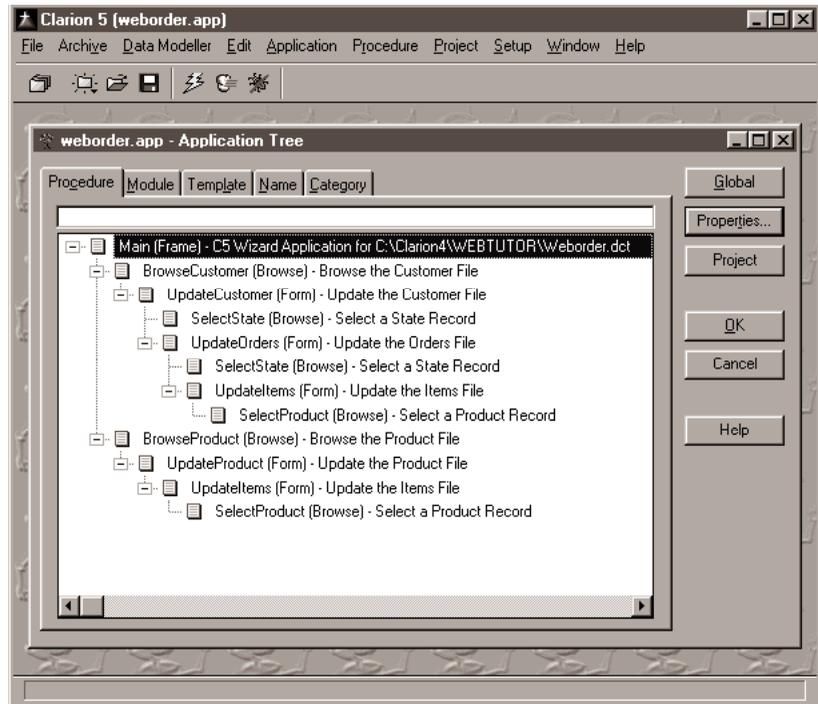
5. Check the **Create an Internet enabled application** box, then press the **Next** button.

This step makes two changes to the application the Application Wizard creates: 1) it makes it 32-bit and 2) it adds the Internet extension templates to the application.



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

The Application Wizard creates the application.



Make the Application

7. Choose **Project** ► **Make** (or press the *Make* button on the toolbar).
Congratulations! Your first Web application is ready to deploy and run.
8. 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:\Clarion5\Examples\Webtutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

Keep in mind that merely dragging files in Explorer creates a shortcut to executable files. If you use the drag-and-drop method, you should right-drag and select copy from the popup menu.

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

3. Copy the files listed below from the *C:\Clarion5\BIN* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

C5RUNx.DLL
C5TPSx.DLL
C5ASCx.DLL
C5DOSx.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 under Windows 95 (on your development machine) because these files are in your PATH. However, NT server 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 *CWBroker1.exe* (or *CWBroker.exe* if you have the full version of the Application Broker) in the *C:\CWICWEB* directory.

Note: For this tutorial, we will use the executable version of the Application Broker. The ISAPI version works in a similar manner, with a few differences. These are discussed in the Application Broker chapter.

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`

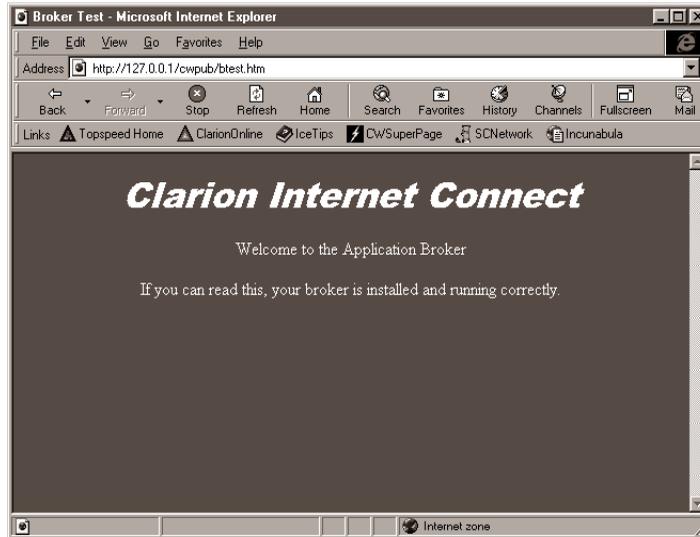
then press ENTER.

Note: If you have the broker set to a port other than 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`



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.

Note: When using the ISAPI version of the broker, you would use this URL to start the test page:

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

or

`http://yourdomain.com/cwpub/btest.htm`

Next, start the application in the browser:

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

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

or

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

then press ENTER.

Note: If you have the broker set to a port other than 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.



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.

℘ 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 four procedures with category—*Browse*.

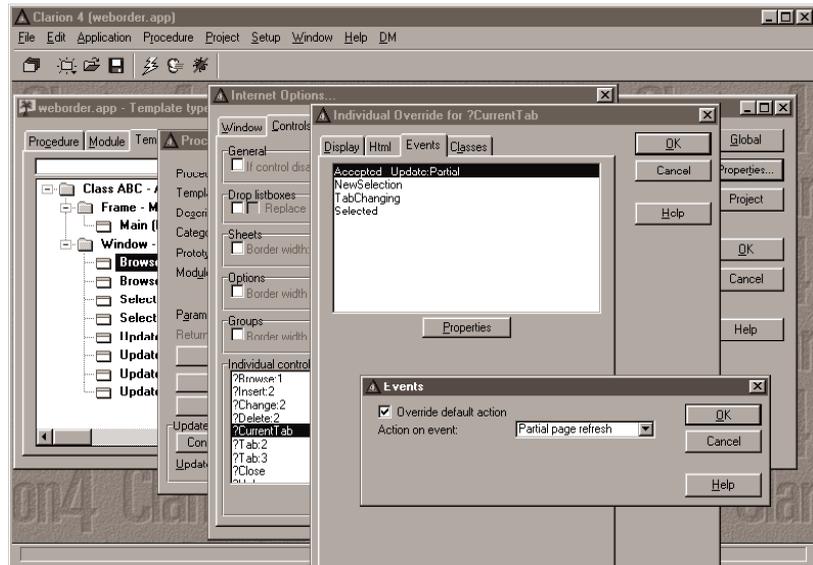
2. Highlight the *BrowseCustomer* procedure, then press the **Properties** button.

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.



2. Press the **OK** buttons on all the windows until you return to the application tree (4 times).
3. Repeat these steps for the three other Browse procedures.
4. Choose **Project ► Make** (or press the *Make* 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:\Clarion5\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. This section is designed to show you how to use the template interface to create the look-and-feel you want.

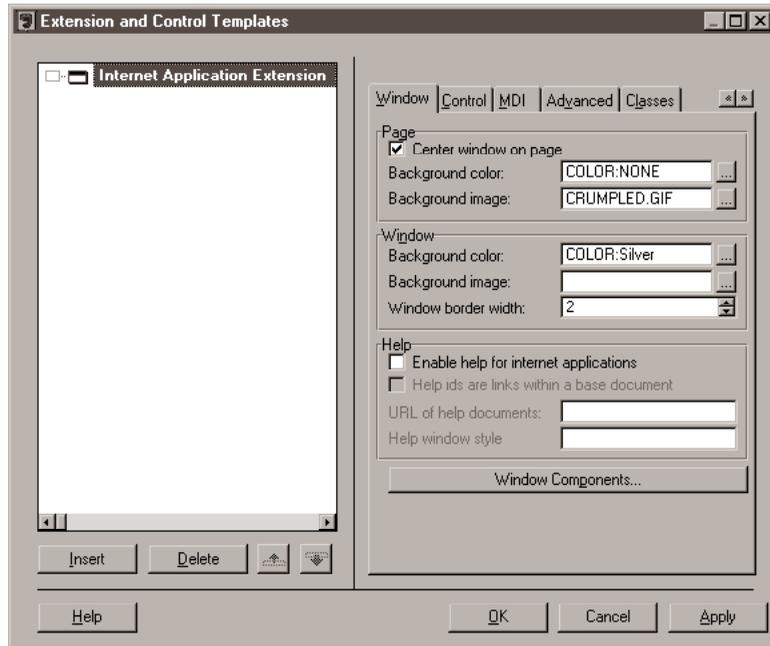
Internet Application Extension Template

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.



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* 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:\Clarion5\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.
4. Copy *Crumpled.gif* from the *C:\Clarion5\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.

TUTORIAL—WEB-ENABLING AN EXISTING APPLICATION

5

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:\Clarion5 and the Application broker in C:\CWICWEB. If you used a different directory, you will have to modify the instructions accordingly.

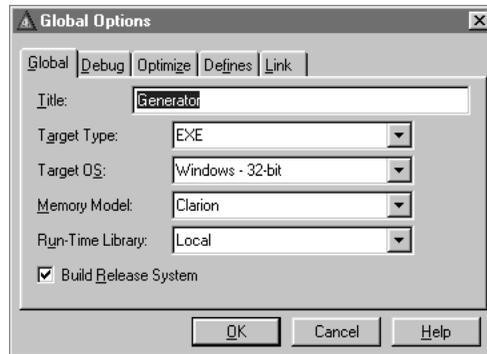
Web-enabling a Clarion application

1. On the **Pick** dialog, press the **Open...** button.
This opens the **Open** dialog.
2. Select the *Application* tab.
3. Select the *C:\Clarion5\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** 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.

Change to 32-bit

1. In the Application Tree, press the **Project** button.
This opens the **Project Editor** window.
2. Press the **Properties** button.
This opens the **Global Options** dialog.
3. In the **Target OS** field, select *Windows - 32-bit*.

Web-enabled applications must be 32-bit.



4. Press the **OK** button on the **Global Options** and the **Project Editor** windows.

That's all it takes to Web-enable an existing application!

Make and Deploy

1. Choose **Project** ► **Make** (or press the *Make* 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:\Clarion5\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.
5. Copy all the icon files (*.ICO) from the *C:\Clarion5\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 *CWBrokr1.exe* (or *CWBroker.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.

7. 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.

7. Highlight the *BrowseCustomer* procedure, then press the **Properties** 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

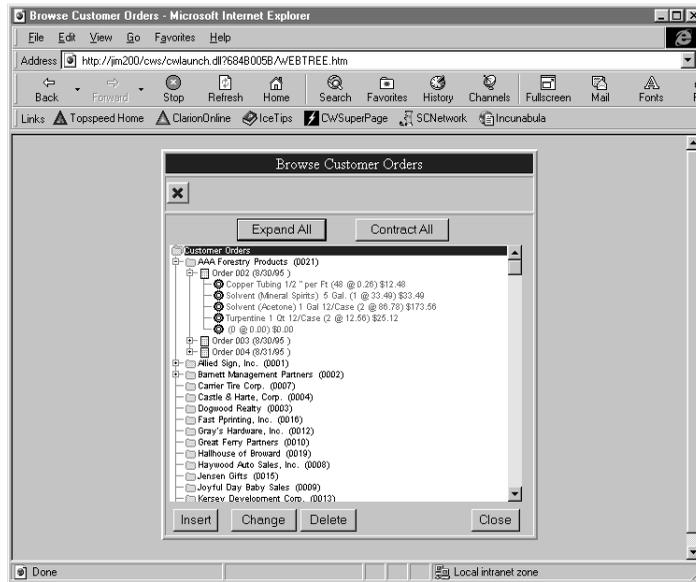
1. Choose **Project ► Make** (or press the *Make* button on the toolbar).
Your Web application is ready to deploy once again.
2. Press the **OK** button on the compiler window.
3. Open Windows Explorer (or Windows NT Explorer).
4. Copy *Weborder.exe* from the *C:\Clarion5\Examples\WebTutor* directory 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.



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.

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:\Clarion5 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

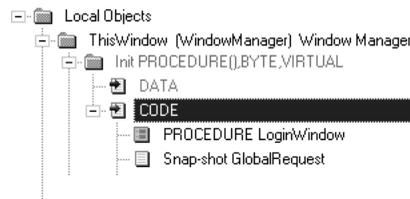
1. In the Application Tree, highlight the *Main* procedure, then press the **Properties** button.

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:



This point ensure 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.

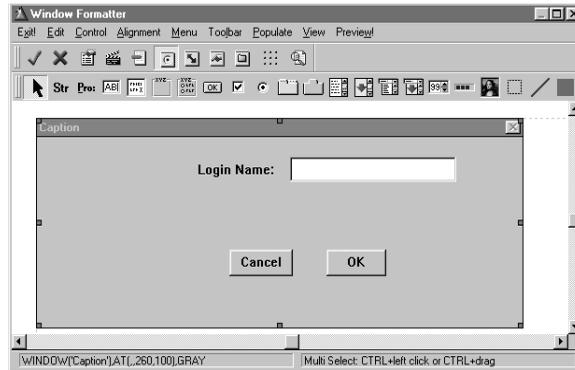
This adds the *LoginWindow* procedure as a *ToDo* item.

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. 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 *Window*, then press the **OK** button.
This opens the **Window Formatter**.

Design the login window

1. Select **Populate ► Field**.
This opens the **File Schematic** dialog.
2. In the **Files** list on the left, highlight *Global Data*, then in the **Fields** 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*.

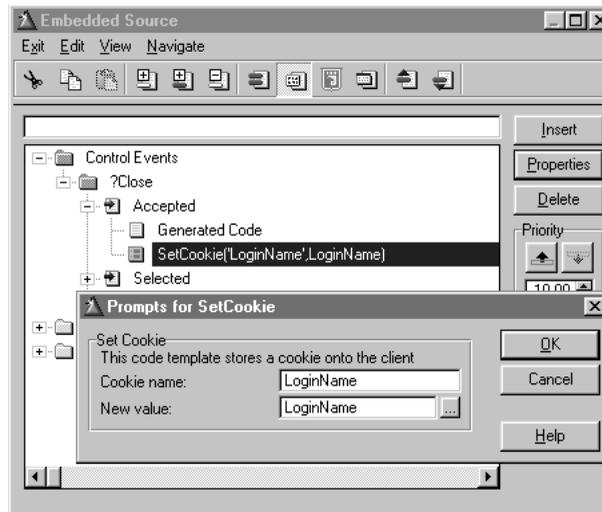


7. Reposition the controls on the window as you see fit.

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*, *Generated Code* embed point then press the **Insert** button.

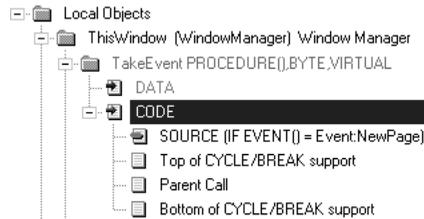
This inserts the code *after* any generated code for the control.
3. Select the *SetCookie* code template then press the **Select** button.
4. In the **Cookie name** field, type *LoginName*.
5. In the **New Value** field, type *LoginName* (or select the *LoginName* global variable from the File schematic 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.



3. Highlight *Source* then press the **Select** button.

4. Type in the source code below:

```

IF EVENT() = Event:NewPage           !If the Web page new
  LoginName = Broker.Http.GetCookie('LoginName') !Get the cookie
  DISPLAY
END
IF LoginName                         ! If value was set
  POST(Event:CloseWindow)           ! close the window
END

```

This code “gets” a cookie when the window is active. If it successfully 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* 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:\Clarion5\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

Run the application

1. Start the Application Broker by DOUBLE-CLICKING on *CWBrokr1.exe* (or *CWBroker.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. 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** button.

This opens the **Procedure Properties** window.

2. Press the **Embeds** button.

This opens the **Embedded Source** window.

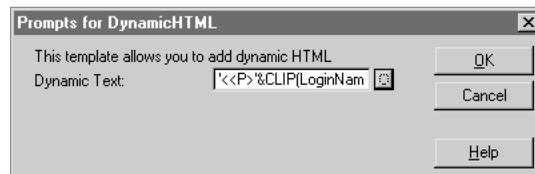
3. Highlight the *Internet-Before the Closing </BODY>* tag embed point, then press the **Insert** button.

This opens the **Select Embed Type** window.

4. Highlight *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* 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:\Clarion5\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@topspeed.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* 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:\Clarion5\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.WriteLine` 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** 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.WriteLine(CLIP(Str1") & 'topspeed' & CLIP(Str2") & SELF.Files.GetAlias('1.GIF') & Str3")  
  Target.WriteLine(CLIP(Str1") & 'icetips' & CLIP(Str2") & SELF.Files.GetAlias('2.GIF') & Str3")  
  Target.WriteLine(CLIP(Str1") & 'finatics' & CLIP(Str2") & SELF.Files.GetAlias('3.GIF') & Str3")  
  Target.WriteLine(CLIP(Str1") & 'flpanthers' & CLIP(Str2") & SELF.Files.GetAlias('4.GIF') & Str3")  
  Target.WriteLine(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

7. Choose **Project ► Make** (or press the *Make* 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:\Clarion5\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.
5. Copy the GIF files (*.gif) from the *C:\Clarion5\Examples\WebTutor* directory to the *C:\CWICWEB\Public* directory.

Examine the application

7. 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.

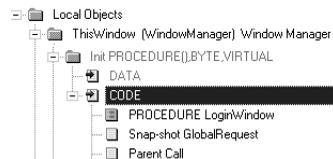
1. In the Application Tree, highlight the *Main* procedure, then press the **Properties** button.

This opens the **Procedure Properties** window.

2. Press the **Embeds** button.

This opens the **Embedded Source** window.

3. Highlight the embed point as shown below:



4. Press the **Insert** button.

This opens the **Select Embed Type** window.

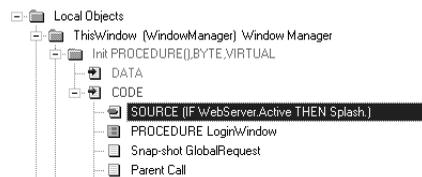
5. Highlight *Source*, then press the **Select** button.

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.



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 **Procedure** window.

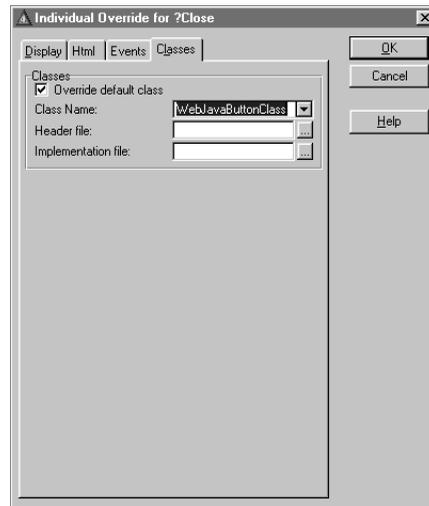
10. Highlight *Splash*, then press the **OK** button.

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** 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.



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 it 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.

7. 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

7. Choose **Project ► Make** (or press the *Make* 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:\Clarion5\Examples\WebTutor* directory to the *C:\CWICWEB\EXEC\WebTutor* directory.

Examine the application

7. 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

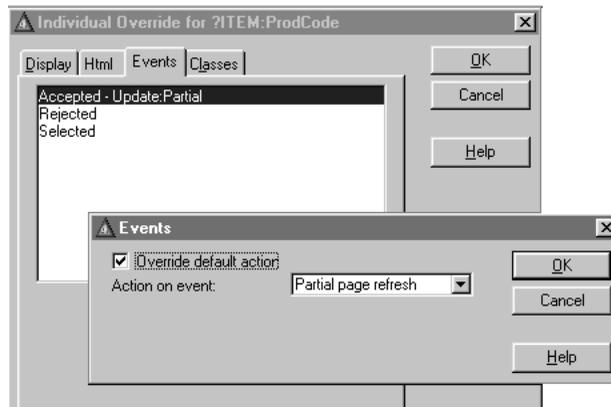
In Windows applications, programmers often embed code to update one control when the value of another control changes. For example, you might embed code to change the total of a line item when the quantity of items changes. The Webtree application has code like this in the *UpdateItems* procedure. The 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** 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).

2. 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* 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:\Clarion5\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 items (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 *UpdateProducts* procedure, then press the **Properties** button.

This opens the **Procedure Properties** window.

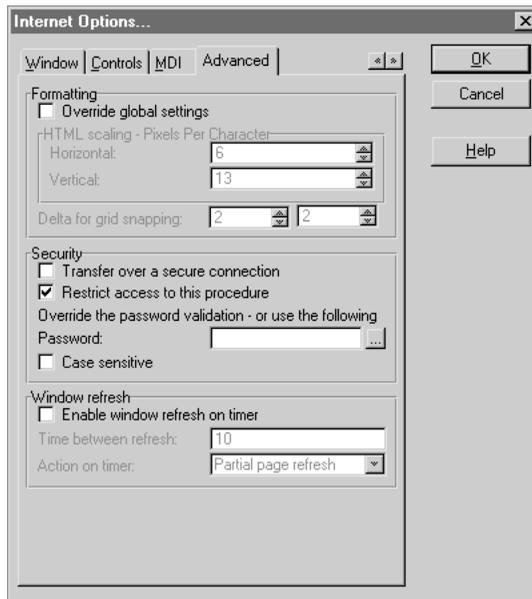
2. Press the **Files** button.

This opens the **File Schematic** 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 **File Schematic** 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.



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* 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:\Clarion5\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 *BrowseProducts* procedure, then press the Properties 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* 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:\Clarion5\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 enabled.
9. Exit the application.

Congratulations! You have successfully 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.

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 `COLORIALOG` 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 `FILEIALOG` 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 `COLORIALOG` 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 filenames 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 `COLORIALOG` 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 `FILEIALOG` 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 COLORIALOG 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 FILEIALOG 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 FILEIALOG 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 `COLORIALOG` 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 `FILEIALOG` 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 `COLORIALOG` 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 `FILEIALOG` 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:

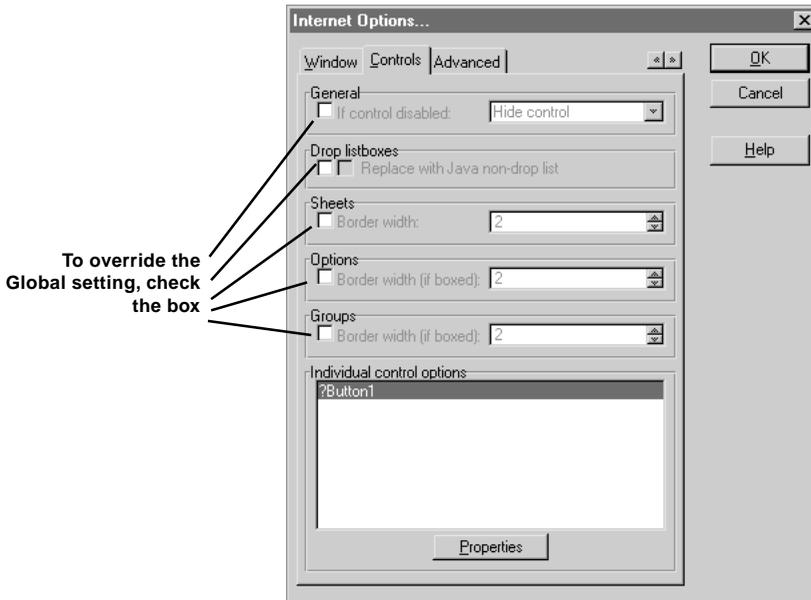
Hide Hides any disabled controls (the default).

Hide if cannot disable

Hides any disabled control when it cannot be disabled on the Web page. Most HTML controls cannot be 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

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.

Note: This feature requires installation of the secure version of the Application Broker. See the *Application Broker* chapter.

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 independent 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 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.

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 suppress 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 not 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 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 COLORIALOG 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 FILEIALOG 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 FILEIALOG 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 COLORIALOG 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.WriteLine` 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.WriteLine` 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*

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 SHEET's behavior for the example above, follow these steps:

1. From the **Procedure Properties** window, press the **Internet Option** 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.

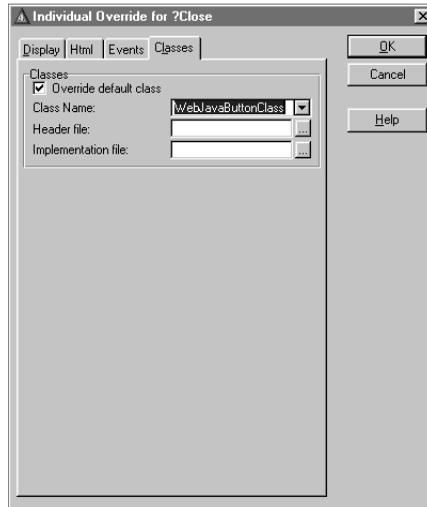
Covering the Download with a Splash Window

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 contain 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** 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

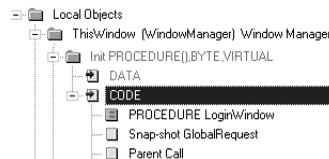
7. In the Application Tree, highlight the *Main* procedure, then press the **Properties** button.

This opens the **Procedure Properties** window.

2. Press the **Embeds** button.

This opens the **Embedded Source** window.

3. Highlight the embed point as shown below:



4. Press the **Insert** button.

This opens the **Select Embed Type** window.

5. Highlight *Source*, then press the **Select** button.

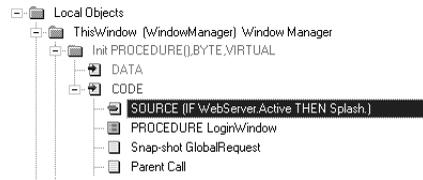
6. In the **Embedded Source** editor, type the following source code:

```
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 **Procedure** 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 `` 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 ``. 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.WriteLine('<<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:

7. Select the Browse procedure, then press the Properties button.

2. In the UpdateButton section of the **Procedure Properties** window, check the **Use Edit in Place** box.

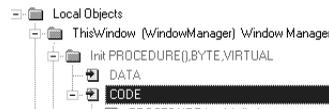
Notice that an update procedure is already specified. Make sure to leave that procedure named.

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. 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 (in this example it is the `WebHtmlStringClass`). 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` 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 several method 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.
- ◆ Transmitting over a (SSL) secure connection.

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.

The third method—transmitting over a secure connection—serves a different purpose. It is not intended to restrict access to a user. It is intended to restrict interception of data during transmission. This security measure can be used alone or in conjunction with either of the other two security measures.

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 a Secure Socket Layer (SSL)

This security measure requires that you run the ISAPI version of the Application Broker under an ISAPI-compliant Web Server and have a Digital Certificate installed. See *The Application Broker* chapter. The Internet Procedure Extension template's SSL support uses the ISAPI SSL encryption for the duration of the procedure. When a procedure with SSL enabled is called, the Application Broker switches into SSL mode. When the procedure terminates, normal access is restored. 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. Allowing you to encrypt only those procedures which need secure transmission improves performance on both the client and server side by utilizing encryption only when it is needed.

To enable SSL for a procedure:

1. From the **Procedure Properties** window, press **Internet Options**.
2. Select the **Advanced** tab
3. Check the **Transfer over a secure connection** box.

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.WriteLine` 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.WriteLine` 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.WriteLine` 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.WriteLine` 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.WriteLine('<<p>Copyright 1997, TopSpeed&trade; Corporation, 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.WriteLine('<<A HREF="http://www.topspeed.com">Visit Topspeed<</A>')
  Target.WriteLine('<<A HREF="http://www.clariononline.com">Visit ClarionOnline<</A>')
  Target.WriteLine('<<A HREF="http://www.icetips.com">Visit IceTips<</A>')
  Target.WriteLine('<<A HREF="http://www.finatics.com">Visit the Finatics<</A>')
  Target.WriteLine('<<A HREF="http://www.topspeed.com/tsnews.htm">TopSpeed 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 ``.

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.WriteLine('<<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.WriteLine('<<IMG SRC="' & SELF.FILES.GETAlias('mygif.gif') & "'>')  
Target.WriteLine('<<applet '  
Target.WriteLine('Codebase = "' & SELF.FILES.GETAlias() & "' '  
Target.WriteLine('code="TickerTape.class">')  
Target.WriteLine('<</applet>')
```

Note: In an APPLET HTML tag, 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
<h2 align="center"><a name="SecondWindowID">Help For Second Window</a></
h2>
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 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. For more information see Images.

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 .

BOX

Not supported--use Embedded HTML to display an image .

ELLIPSE

Not supported--use Embedded HTML to display an image .

ENTRY	Created as an HTML entry field <code><INPUT TYPE=TEXT VALUE = value in field ></code> . Entry patterns are not supported.
BUTTON	Created as an <code><INPUT TYPE=SUBMIT ></code> 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 <code>/PUBLIC</code> directory.
PROMPT	Displays as text.
OPTION	Created as an HTML <code><OPTION></code> . If an OPTION has the BOXED attribute, then it is implemented in HTML as a <code><TABLE></code> with the border specified in the Global or Procedure options for OPTIONS .
CHECK	Created as an HTML checkbox <code><INPUT TYPE=CHECKBOX VALUE = value in field ></code>
GROUP	If a GROUP has the BOXED attribute, then it is implemented in HTML as a <code><TABLE></code> 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 <code>/PUBLIC</code> 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 <code>/PUBLIC</code> directory.
FileDropCombo	Created as an HTML drop-down (<code><SELECT></code> 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 (<code><SELECT></code> 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 <code><INPUT TYPE=TEXT VALUE = value in field ></code>
COMBO	Created as an HTML entry field <code><INPUT TYPE=TEXT VALUE = value in field ></code> .
SPIN	Created as an HTML entry field <code><INPUT TYPE=TEXT VALUE = value in field ></code> .
TEXT	Created as an HTML Text field <code><TEXTAREA ></code> .
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 <code><TABLE ></code> inside an HTML page.
WINDOW	HTML <code><TABLE</code> 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 borderwithd to provide a similar appearance.
TOOLBAR	Created as a row in an HTML <code><TABLE></code> . 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 Js1EventsClass                               !Declare JavaEvents object
WebServer WebServerClass                                !Declare WebServer object
WebFileManager WebFilesClass                           !Declare WebFileManager 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
WebFileManager.Init(1, '')                              !Initialize WebFileManager
JavaEvents.Init                                        !Initialize JavaEvents
Broker.Init('HelloWeb', WebFileManager)                !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
!Usual Windows event handling
!Close window on ?Bye button
  IF EVENT() = EVENT:Accepted
    POST(Event:CloseWindow)
  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 *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

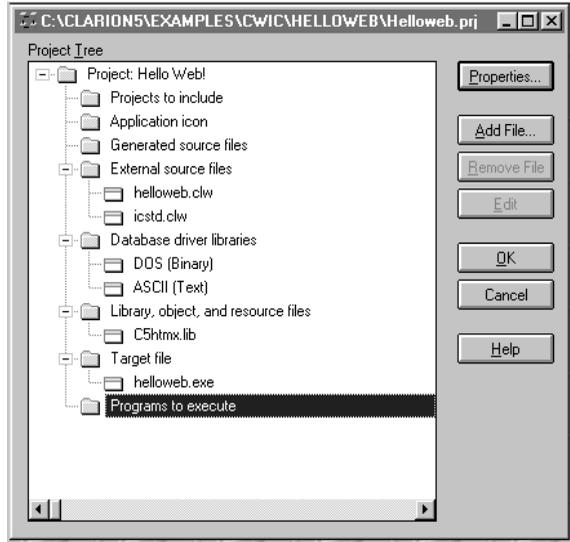
The IBC Library objects use the ASCII Database Driver to process reports. You must add the ASCII driver to the **Database driver libraries** branch of the Project tree to resolve IBC references to ASCII driver procedures.

C5HTMx.LIB

C5HTMx.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 C4HTMx.LIB file to the **Library, object, and resource files** branch of the project tree.

Web-enabled Programs Must be 32-bit

To have any practical benefits, web-enabled programs must be 32-bit. This is because, in an internet environment, multiple clients may request the application at the same time; therefore, the program must support multiple instances on the web server. Unlike 16-bit programs, 32-bit programs allow multiple instances.



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 Internet Connect installation CD.

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.

<u>Internet Builder Class</u>	<u>Template Generated Object</u>
BrokerClass	Broker
HtmlClass	HtmlManager
JsIEventsClass	JavaEvents
TextOutputClass	Target
HttpClass	Broker.Http
WebFilesClass	WebFilesManager, Broker.Files, HtmlManager.Files, Broker.Http.Files, JavaEvents.Files, WebServer.Files, WebWindow.Files, and Target.Files
WebServerClass	WebServer
WebClientManagerClass	Broker.CurClient
WebFrameClass	MainFrame <i>and</i> 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)

JsIEventsClass (JavaEvents)

Init (initialize the JsIEventsClass object)
Kill (shut down the JsIEventsClass object)

TextOutputClass (HtmlManager or Target)

Writeln (write one line of text)

HttpClass (Broker.Http)

GetCookie (get cookie from client)
SetCookie (set cookie from client)
SetProcName (set protected area name)
SetProgName (set server name)

WebFilesClass (WebFileManager 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)
Freq (control number)
ParentFreq (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 button.

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 ratio 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 single-connect version of the Application Broker 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 variety 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

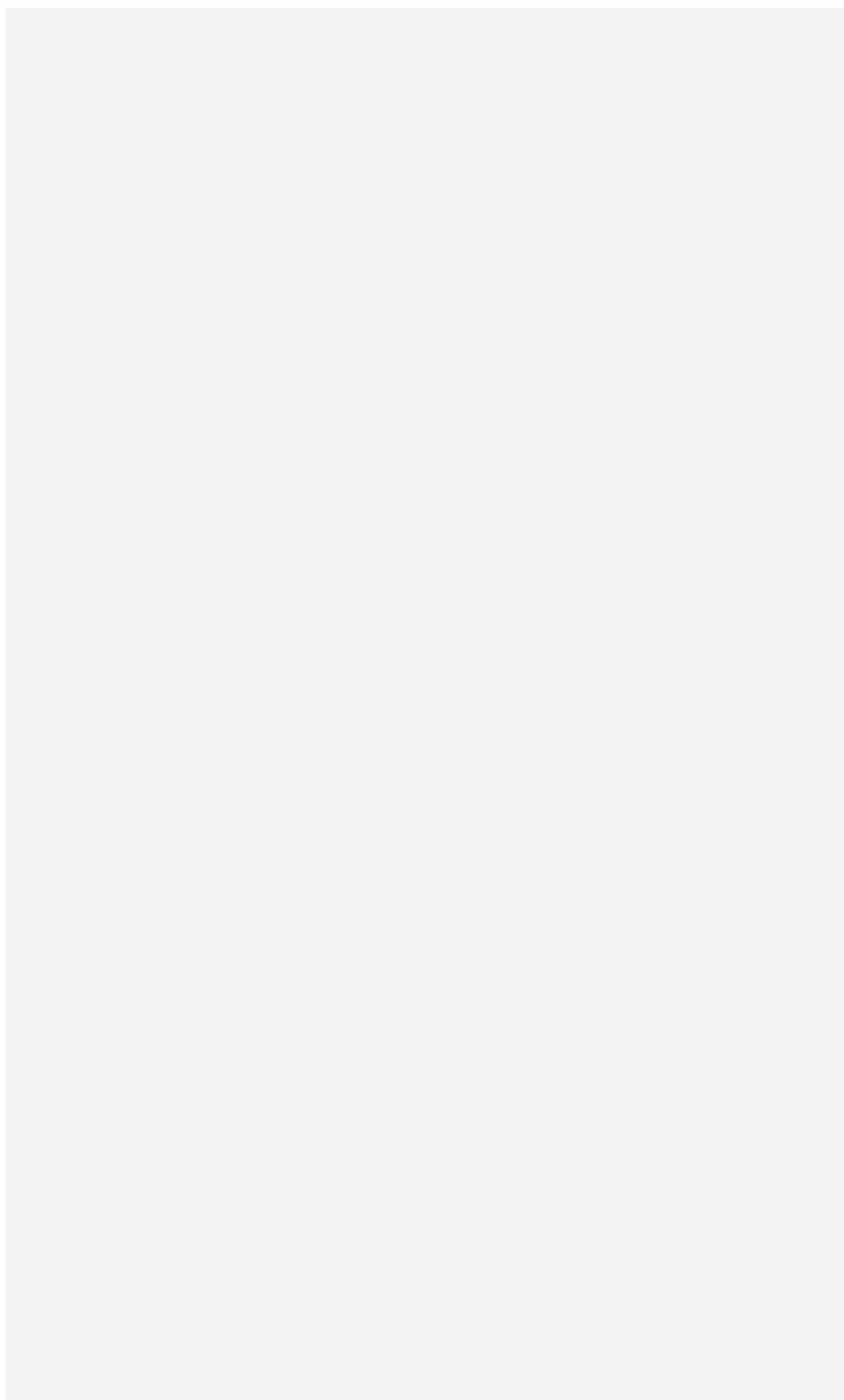
(Clarion) The Session Router distributes remote computing sessions to multiple Application Brokers over the Internet, when high popularity or demand requires the deployment of additional Internet servers. The Session Router is available separately.

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

Symbols

*.class	27
\admin	24
\exec	24
\public	24
\scripts	24
\sec_scr	24
\secure	24
127.0.0.1	35

A

Accepted	68
Action on Event	95
Active	
WebServerClass	136
AddControl	
WebWindowClass	137
AddServerProperty Code Template	105
Alignment	82, 83, 97, 98
WebCaptionClass	138
Allow dynamic updates	93
AllowJava	
WebWindowBaseClass	136
ALT	94
applet	139
Application Toolbar	101
Application Broker	19, 139
Application Broker Setup	29
Application Broker's setup utility	27
Application Menu	101
AuthorizeArea	
WebWindowClass	137
Autospot Hyperlinks	93

B

Background color ..	76, 82, 83, 84, 85, 86, 87, 97, 98, 99
Background image	
76, 77, 82, 83, 84, 85, 86, 87, 97, 98, 99, 100	
BorderWidth	
WebWindowBaseClass	136
Broker	134, 139
Broker.CurClient	134
Broker.CurClient.IP	135
Broker.Http	134

Broker.Http.GetCookie	135
Broker.Http.SetCookie	135
Broker.Http.SetProcName	135
Broker.Http.SetProgName	135
Broker.Init	135
Broker.Kill	135
Broker.ServerName	135
BrokerClass	133, 135

C

Caption	82, 97
Center	76
Center Window on Page	76, 86
centering an IMAGE	67
CLARION.CAB	27
CLARION.ZIP	27
Class Overrides	85
Classes Local to Application Broker	81
Client	139
Client Area	85
Close button	84, 99
CloselImage	
WebWindowBaseClass	136
CommandLine	
WebServerClass	136
Complete page refresh	95
concurrency	20
Connect	
WebServerClass	136
Control	78
Control options	88
Control Overrides	93
cookie	139
Cookies	104
CopyControlsToWindow	
WebFrameClass	136
Create Application dialog	38, 50
Create extra close button	84
CreateCaption	
WebWindowBaseClass	136
CreateClose	
WebWindowBaseClass	136
CreateHtml	
WebControlClass	137
CreateHtmlPage	
WebWindowClass	137
CurClient	134
CWBroker.exe	20
CWBrokr1.exe	20
CWISAPI.DLL	27
CWISSET.TXT	25
CWLAUNCH.DLL	27
CWSECURE.DLL	27

D	
default button	139
Default Home Page	29
Delta for grid snapping	80, 90
deploy	31
Digital Certificate	24
Directories	28
disabled	139
DisabledAction	
WebControlClass	137
WebWindowBaseClass	136
Downloading the Java Support Library	65
Drop listboxes	78
Dynamic HTML Code Template	103
dynamic updates	93
E	
Edit-In-Place	73
Enable	32
Enable Help for internet applications	77
Enable Refresh on timer	92
encryption	23, 139
Event Handling	68
F	
Feq	
WebControlClass	137
Firewalls	22
Font	139
Font color	83, 98
Font family name	82, 97
Font size	82, 97
font style	139
FormatBorderWidth	
WebWindowBaseClass	136
Formatting	90
fourth generation language	9
Frame Menu	79
Frame Toolbar	80
FrameWindow	
WebFrameClass	136
G	
GetAlias	
WebFilesClass	135
GetControllInfo	
WebWindowClass	137
GetCookie	
HttpClass	135
GetCookie Code Template	104
GETINI	104
GetServerProperty Code Template	105
GetToolBarMode	
WebWindowClass	137
GIF image	140
Global Internet Application Extension	75
Group Border width	79
GroupBorderWidth	
WebWindowBaseClass	136
H	
hard disk space	15
Help	77, 87
Help Document	78, 88
Help Ids are links within a base document	77, 88
Help Ids are links within a base document box	88
Help Window Style	77, 88
HelpDocument	
WebWindowBaseClass Properties	137
HelpEnabled	
WebWindowBaseClass Properties	137
HelpRelative	
WebWindowBaseClass Properties	137
hide	140
Horizontal Pixels per Char	80, 90
HTML	140
HtmlClass	135
HtmlManager	134
HtmlManager.Init	135
HtmlManager.Kill	135
HtmlPreview	134, 138. See WebReportClass
HtmlPreview.Init	138
HtmlPreview.Kill	138
HtmlPreview.Preview	138
HTTP	140
HTTP header	105
Hybrid Web/Windows Application	140
Hyperlinks	22, 93
I	
IBC templates	75
IC:CurControl	134, 137
IC:CurControl.CreateHtml	137
IC:CurControl.DisabledAction	137
IC:CurControl.Feq	137
IC:CurControl.ParentFeq	137
IC:CurControl.ResetFromQueue	138
IC:CurControl.SetAutoSpotLink	137, 138
IC:CurControl.SetBorderWidth	137

- IC:CurControl.SetDescription 137
 - IC:CurControl.SetEventAction 138
 - IC:CurControl.SetQueue 138
 - IC:CurFrame 134
 - IC:CurFrame.CopyControlsToWindow 136
 - IC:CurFrame.FrameWindow 136
 - IC:CurFrame.TakeEvent 136
 - Icon 140
 - If control disabled 78
 - IIS 23
 - IIS Internet Service Manager 25
 - Image for close 84
 - Implementation file 85
 - Include caption 97
 - include file 140
 - Individual Overrides for a Control 93
 - Init
 - BrokerClass 135
 - HtmlClass 135
 - JslEventsClass 135
 - WebControlClass 137
 - WebFilesClass 135
 - WebReportClass 138
 - WebServerClass 136
 - WebWindowClass 137
 - Internet Builder Class Templates 75
 - Internet Developer's Kit 140
 - Internet Information Server 23
 - Internet Server Application Programming Interface 23
 - ISAPI 20
 - IsSecure
 - WebWindowClass 137
- J**
- Java Support Library (JSL) 140
 - JavaEvents 134. *See* JslEventsClass
 - JavaEvents.Init 135
 - JavaEvents.Kill 135
 - JavaLibraryPath
 - WebServerClass 136
 - JPG 140
 - JSL 140
 - JSL data 140
 - JslEventsClass 135
- K**
- Kill
 - BrokerClass 135
 - HtmlClass 135
 - JslEventsClass 135
 - WebControlClass 137
 - WebFilesClass 135
 - WebReportClass 138
 - WebServerClass 136
 - WebWindowClass 137
- L**
- localhost 35
 - Log File Name 29
- M**
- MainFrame 134
 - MainFrame.CopyControlsToWindow 136
 - MainFrame.FrameWindow 136
 - MainFrame.TakeEvent 136
 - Max Simultaneous Connections 29
 - MDI 79, 90
 - Menu 98
 - MenubarType
 - WebWindowBaseClass 136
 - WebWindowClass 137
- O**
- Option Border width 79
 - OptionBorderWidth
 - WebWindowBaseClass 136
 - Override Global settings 86, 87
 - Overrides for a Control 93
- P**
- Page Settings 86
 - Page to return to on exit 80
 - PageToReturnTo
 - WebServerClass 136
 - ParentFeq
 - WebControlClass 137
 - Partial page refresh 95
 - Partial Refresh
 - Updating Controls 68
 - password 29, 91
 - Personal Web Server 23
 - PING 36
 - port 80 20
 - Port Number 29
 - Preview
 - WebReportClass 138
 - Process on Browser 95
 - product registration 13
 - programming language 9

ProgramName		SetFormatOptions	
WebServerClass	136	WebWindowClass	137
proxy servers	22	SetNewPageDisable	
Public Directory	29	WebServerClass	136
PUTINI	104	SetPageBackground	
Q		WebWindowClass	137
Quit		SetPassword	
WebServerClass	136	WebWindowClass	137
R		SetProcName	
RAM	15	HttpClass	135
Refresh on timer	92	SetProgName	
Remote Access to the Application Broker	31	HttpClass	135
Remote Computing Session	141	SetQueue	
Remote password	29	WebJavaListClass	138
ResetFromQueue		SetSendWholePage	
WebJavaListClass	138	WebServerClass	136
Restrict Access to this procedure	91	SetSplash	
Return if launched from browser	88	WebWindowClass	137
Reusable Client	141	SetTimer	
		WebWindowClass	137
		Setup	31
		Sheet Border width	78
		SheetBorderWidth	
		WebWindowBaseClass	136
		single-connection Application Broker	20
		SSL	23, 91
		Static HTML Code Template	103
		Status	32
		Sub directory for pages	81
		SuppressControl	
		WebWindowClass	137
		Suspend	31
		T	
		TakeEvent	
		WebFrameClass	136
		WebWindowClass	137
		Target	134
		Target.GetAlias	135
		Target.Writeln	135
		test locally	35
		TextOutputClass	135
		Time out	81
		Timeout	
		WebServerClass	136
		TIMER	92
		Timer	141
		Toolbar	98, 140
		Transfer over a secure connection	91
Secure Socket Layer	23, 91		
Secure Sockets Layer	23		
Security	91		
SelectTarget			
WebFilesClass	135		
Server	141		
Session Router	141		
SetAutoSpotLink			
WebJavaListClass	138		
WebJavaStringClass	137		
SetBackground			
WebCaptionClass	138		
WebClientAreaClass	138		
WebMenuBarClass	138		
WebToolBarClass	138		
WebWindowClass	137		
SetBorderWidth			
WebControlClass	137		
SetCookie			
HttpClass	135		
SetCookie Code Template	104		
SetDescription			
WebHtmlImageClass	137		
SetEventAction			
WebJavaListClass	138		
SetFont			
WebCaptionClass	138		

- U**
- Ultra-thin Reusable Client 141
 - Uniform Resource Location 21
 - Update Image dynamically 94
 - URL 21
 - URL of help documents 77, 87
 - Use Debug Setup 30
 - Use Log File 29
 - Use Long Filenames 81
- V**
- ValidatePassword
 - WebWindowClass 137
 - Vertical Pixels per Char 80, 90
 - virtual root 28
- W**
- Web-enable a Clarion application 75
 - Web/Windows Application 140
 - WebCaption 134, 138
 - WebCaption.Alignment 138
 - WebCaption.SetBackground 138
 - WebCaption.SetFont 138
 - WebCaptionClass 138
 - WebClientArea 134, 138
 - WebClientArea.SetBackground 138
 - WebClientAreaClass 138
 - WebClientManagerClass 135
 - WebControlClass 137
 - WebFiles 135
 - WebFilesClass 135
 - WebFilesManager 134
 - WebFilesManager.GetAlias 135
 - WebFilesManager.Init 135
 - WebFilesManager.Kill 135
 - WebFilesManager.SelectTarget 135
 - WebFrame 136
 - WebFrameClass 136
 - WebHtmlImageClass 137
 - WebJavaListClass 138
 - WebMenubar 134, 138
 - WebMenubar.SetBackground 138
 - WebMenubarClass 138
 - WebReportClass 138
 - WebServer 134
 - WebServer.Active 136
 - WebServer.CommandLine 136
 - WebServer.Connect 136
 - WebServer.Init 136
 - WebServer.JavaLibraryPath 136
 - WebServer.Kill 136
 - WebServer.PageToReturnTo 136
 - WebServer.ProgramName 136
 - WebServer.Quit 136
 - WebServer.SetNewPageDisable 136
 - WebServer.SetSendWholePage 136
 - WebServer.TimeOut 136
 - WebToolBar 134, 138
 - WebToolBar.SetBackground 138
 - WebToolBarClass 138
 - WebWindow 134, 136, 137
 - WebWindow.AddControl 137
 - WebWindow.AllowJava 136
 - WebWindow.AuthorizeArea 137
 - WebWindow.BorderWidth 136
 - WebWindow.CloseImage 136
 - WebWindow.CreateCaption 136
 - WebWindow.CreateClose 136
 - WebWindow.CreateHtmlPage 137
 - WebWindow.DisabledAction 136
 - WebWindow.FormatBorderWidth 136
 - WebWindow.GetControlInfo 137
 - WebWindow.GetToolBarMode 137
 - WebWindow.GroupBorderWidth 136
 - WebWindow.Init 137
 - WebWindow.IsSecure 137
 - WebWindow.Kill 137
 - WebWindow.MenuubarType 136, 137
 - WebWindow.OptionBorderWidth 136
 - WebWindow.SetBackground 137
 - WebWindow.SetFormatOptions 137
 - WebWindow.SetPageBackground 137
 - WebWindow.SetPassword 137
 - WebWindow.SetSplash 137
 - WebWindow.SetTimer 137
 - WebWindow.SheetBorderWidth 136
 - WebWindow.SuppressControl 137
 - WebWindow.TakeEvent 137
 - WebWindow.ValidatePassword 137
 - WebWindowBaseClass 136
 - WebWindowBaseClass Properties
 - HelpDocument 137
 - HelpEnabled 137
 - HelpRelative 137
 - WebWindowClass 137
 - Window border width 77, 87
 - Window Settings 76, 87
 - Winipcfg.exe 35
 - Writeln
 - TextOutputClass 135

