



Web Application Guide

VERSION 2018.0.01

jade

Jade Software Corporation Limited cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages, or loss of profits. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Jade Software Corporation Limited.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Copyright © 2018 Jade Software Corporation Limited.

All rights reserved.

JADE is a trademark of Jade Software Corporation Limited. All trade names referenced are the service mark, trademark, or registered trademark of the respective manufacturer.

For details about other licensing agreements for third-party products, you must read the JADE **ReadMe.txt** file.

Contents

Contents	iii
Before You Begin	vi
Who Should Read this Guide	vi
What's Included in this Guide	vi
Related Documentation	vi
Conventions	vii
Chapter 1 Implementing Web Applications	8
Overview	8
Creating Web Pages	10
Adding Controls to a Web Page	11
Generating HTML	12
Enhanced Web Functionality	12
Application::webMinimumResponseTime Property	13
Application Class Methods	13
Form Class Methods	14
WebSession Class Properties	14
WebSession Class Methods	14
Window::userScript Property	15
Window Class Web-Related Methods	15
Generating a Web Image	16
Handling Web Forms in Your JADE Code	16
Handling Events on Web Pages	16
Multiple Selections in List Boxes on Web Forms	17
Enabling Your JADE Application for HTML Thin Client Access	17
Specifying Your HTML Thin Client Access Options	19
Enabling Event Handling on Web Pages	23
Using a Web Browser to Access Your JADE Application	24
Accessing a JADE Application from an HTML Thin Client	24
Invoking a User Method	24
JADE Connectivity from the Internet	25
Managing Your Web Sessions	26
Initiation and Process Flow	26
JADE Processing of an HTML Client Request	27
Message Logging	27
Suppressing the Logging of Messages	28
Handling Exceptions in an HTML-Enabled Application	28
Message Box Handling	28
Chapter 2 Monitoring Your Web Sessions	29
Overview	29
Opening a Browser for a Web Services Application	30
Shutting Down the Web-Enabled Application	31
Clearing the Displayed Information	31
Specifying the Content of Logged Messages	31
Disabling Logging	31
Directing Web Monitor Information to a File	32
Restarting a Web Session	32
Closing a Session	32
Displaying Session Details	33
Listing Active Web Sessions	33
Displaying Session Statistics	33
Accessing Online Help	34
Displaying Information about Your HTML Thin Client Application	34

Chapter 3

Configuring Web Applications

35

Overview

35

Structure of the Application Configuration File

36

Elements in the Application Configuration File

37

jade_config element

37

application element

37

web_config element

38

connection_name element

38

application_copies element

38

protocol_family

38

session_timeout element

39

minimum_response_time element

39

disable_messages element

39

output_maximum_length element

39

log_file_name element

39

logmessagecontent element

39

disable_logging element

40

lock_retries element

40

prompt_on_shutdown element

40

firewall element

40

monitor_font element

40

base_uri element

40

protocol element

40

machine_name element

41

virtual_directory element

41

support_library element

41

jade_forms element

41

physical_directory element

41

maximum_HTML_size element

41

scrolling_text element

41

show_modal element

41

cross_browser element

42

form_style element

42

use_html4 element

42

web_events element

42

control_name element

42

image_type element

42

page_sequencing element

42

html_documents element

42

home_page element

42

html_page_sequencing element

43

web_services_provider element

43

read_timeout element

43

use_session_handling element

43

Structure of the Web Services Consumer Configuration File

43

Elements in the Web Services Consumer Configuration File

44

jade_config element

44

web_services_consumer element

44

web_config element

45

consumer element

45

endpoint element

45

maximum_connections element

45

connection_timeout element

45

send_timeout element

45

receive_timeout element

45

Creating and Maintaining the Configuration Files

45

Using the Web Configuration Application

45

Web Configuration Application File Menu

47

New Command

47

Application Command

48

Consumer Command	49
Open Command	50
Append New Command	51
Save Command	51
Save As Command	51
Exit Command	52
Web Application Configuration Examples	52
Configuration 1: Minimum	52
Configuration 1 - JadeHttp.ini File	53
Configuration 1 - Web Application Configuration File	53
Configuration 2: Parallel Requests	53
Configuration 2 - JadeHttp.ini File	54
Configuration 2 - Web Application Configuration File	54
Configuration 3: Multiple Connection Groups	55
Configuration 3 - JadeHttp.ini File	55
Configuration 3 - Web Application Configuration File	55
Configuration 4: Multiple JADE Nodes	56
Configuration 4 - JadeHttp.ini File	57
Configuration 4 - Web Application Configuration File	57
Configuration 5: Multiple Application Copies in Multiple Nodes	58
Configuration 5 - JadeHttp.ini File	58
Configuration 5 - Web Application Configuration File	58
Configuration 6: Multiple JADE Servers	59
Configuration 6 - JadeHttp.ini File	60
Configuration 6 - Web Application Configuration File	60
Configuration 7: Multiple Web Servers	61
Configuration 7 - JadeHttp.ini File	61
Configuration 7 - Apache Configuration File	61
Configuration 7 - Web Application Configuration File	62
Chapter 4 Using the Rich Internet Application (RIA) Framework	63
Overview	63
JavaScripts Generated from Web Services	64
Generating JavaScript	64
Asynchronous Responses and Callbacks	66
Client-Side Caching	66
Restricted Cross-Domain Calls	67
JavaScript Object Notation	67
Example of JavaScript to Invoke Web Services	67
Generated JavaScript API for a Web Service	68
WSUtil.js	69
Classes.js	69
Web-service_api.js	72
Callbacks, Preprocessors, and Exception Handlers	73
Web-service_types.js	74
Web-service_testharness.html	74

Before You Begin

The *JADE Web Application Guide* is intended as a main source of information when you are developing, administering, and implementing Web JADE applications; that is, JADE forms, HTML documents, Web services, and REST services.

Who Should Read this Guide

The main audience for the *JADE Web Application Guide* is expected to be system administrators.

What's Included in this Guide

The *JADE Web Application Guide* has four chapters.

Chapter 1	Covers implementing HTML-enabled applications
Chapter 2	Covers monitoring Web applications
Chapter 3	Covers configuring Web applications and Web service consumers
Chapter 4	Covers Rich Internet Application (RIA) framework support

Related Documentation

Other documents that are referred to in this guide, or that may be helpful, are listed in the following table, with an indication of the JADE operation or tasks to which they relate.

Title	Related to...
JADE Database Administration Guide	Administering a JADE database
JADE Developer's Reference	Developing or maintaining JADE applications
JADE Development Environment User's Guide	Using the JADE development environment to development JADE applications
JADE Encyclopaedia of Classes	System classes (Volumes 1 and 2), Window classes (Volume 3)
JADE Encyclopaedia of Primitive Types	Primitive types and global constants
JADE Initialization File Reference	Maintaining JADE initialization file parameter values
JADE Installation and Configuration Guide	Installing and configuring JADE
JADE Report Writer User's Guide	Using the JADE Report Writer to develop and run reports
JADE Runtime Application Guide	Administering JADE deployed runtime applications
JADE Thin Client Guide	Administering JADE thin client environments

Conventions

The *JADE Web Application Guide* uses consistent typographic conventions throughout.

Convention	Description
Arrow bullet (➤)	Step-by-step procedures. You can complete procedural instructions by using either the mouse or the keyboard.
Bold	<p>Items that must be typed exactly as shown. For example, if instructed to type foreach, type all the bold characters exactly as they are printed.</p> <p>File, class, primitive type, method, and property names, menu commands, and dialog controls are also shown in bold type, as well as literal values stored, tested for, and sent by JADE instructions.</p>
<i>Italic</i>	<p>Parameter values or placeholders for information that must be provided; for example, if instructed to enter <i>class-name</i>, type the actual name of the class instead of the word or words shown in italic type.</p> <p>Italic type also signals a new term. An explanation accompanies the italicized type.</p> <p>Document titles and status and error messages are also shown in italic type.</p>
Blue text	Enables you to click anywhere on the cross-reference text (the cursor symbol changes from an open hand to a hand with the index finger extended) to take you straight to that topic. For example, click on the " Handling Events on Web Pages " cross-reference to display that topic.
Bracket symbols ([])	Indicate optional items.
Vertical bar ()	Separates alternative items.
Monospaced font	Syntax, code examples, and error and status message text.
ALL CAPITALS	Directory names, commands, and acronyms.
SMALL CAPITALS	Keyboard keys.

Key combinations and key sequences appear as follows.

Convention	Description
KEY1+KEY2	Press and hold down the first key and then press the second key. For example, "press Shift+F2" means to press and hold down the Shift key and press the F2 key. Then release both keys.
KEY1,KEY2	Press and release the first key, then press and release the second key. For example, "press Alt+F,X" means to hold down the Alt key, press the F key, and then release both keys before pressing and releasing the X key.

This chapter covers the following topics.

- [Overview](#)
- [Creating Web Pages](#)
 - [Adding Controls to a Web Page](#)
- [Generating HTML](#)
 - [Enhanced Web Functionality](#)
 - [Generating a Web Image](#)
 - [Handling Web Forms in Your JADE Code](#)
 - [Handling Events on Web Pages](#)
 - [Multiple Selections in List Boxes on Web Forms](#)
- [Enabling Your JADE Application for HTML Thin Client Access](#)
 - [Specifying Your HTML Thin Client Access Options](#)
- [Enabling Event Handling on Web Pages](#)
- [Using a Web Browser to Access Your JADE Application](#)
- [Accessing a JADE Application from an HTML Thin Client](#)
- [Invoking a User Method](#)
- [JADE Connectivity from the Internet](#)
- [Managing Your Web Sessions](#)
 - [Initiation and Process Flow](#)
- [JADE Processing of an HTML Client Request](#)
- [Message Logging](#)
- [Handling Exceptions in an HTML-Enabled Application](#)
 - [Message Box Handling](#)

Overview

The HTML thin client mode enables you to use the Internet to access your JADE applications. Web browsers, such as Microsoft Explorer, provide a convenient client interface to JADE data on distributed servers.

The HTML thin client mode provides the following features.

- Automatic generation of the Web interface

The JADE development environment enables you to create Web pages. You can create a JADE application in one complete graphical environment, and deploy this application on the Web. HyperText Markup Language (HTML) code is generated automatically.

- Session management

JADE creates a session for each user who accesses a JADE application from an HTML thin client. The TCP/IP communications protocol is used to access JADE applications from the Internet through a Web server.

- Import of external HTML files

JADE enables you to import external HTML files by using a JADE development environment wizard. Corresponding classes and properties are created for each HTML file that is imported. For details, see "[Adding and Maintaining HTML Documents](#)", in Chapter 12 of the *JADE Development Environment User's Guide*.

- Web browsers

The dynamically generated HTML code enables you to generate a single user interface that is compatible with multiple browsers (that is, with any browser that is compliant with HTML 3.2).

- Web server

The Microsoft Internet Information Server (IIS) or the Apache HTTP Server is used as the Web server.

The HTTP or HTTPS communication protocol is used to send and receive messages between the Internet and a Web server for Web services, REST services, and Web-enabled applications.

The connection between the JADE Web application and the **jadehttp** module can be over TCP/IP or a named pipe. When running JADE in HTML thin client mode using a named pipe, your JADE application and IIS must reside on the same workstation. When using a TCP/IP connection, the machine hosting IIS or Apache HTTP Server can be different from the machine that is running the JADE application, to provide greater security via firewalls.

If you are uploading images from another machine for a JADE Web-enabled form over a TCP/IP connection, the **Firewall** parameter in the [[Jadehttp Files](#)] section of the JadeHttp initialization file or the **Firewall** configuration directive in the JADE **mod_jadehttp** module *and* the **Firewall** parameter in the [[WebOptions](#)] section of the JADE initialization file must both be set to **true**.

- Direct Web services

Use the TCP communication protocol for *direct* Web services.

Notes The TCP protocol for direct Web services communication works only between JADE systems.

For direct Web services, the machine name must contain the machine name or the IP address followed by a colon (:) character then by a TCP port number on which this service is offered.

- JADE clients

The user interface for a standard (fat) JADE client can be the same as that defined for an HTML thin client, or it can be different, enabling you to have one object model but with different views of that model. The JADE clients can run Web-enabled applications, Web services, and REST services.

- Security

You can implement security using:

- Operating system security and IIS or Apache HTTP Server security for data access
- Secure Sockets Layer (SSL) for data transmission

For details, see "[Internet Access Support](#)", in Chapter 2 of the *JADE Object Manager Guide*.

To ensure that an application specified in a Web browser cannot cause an attachment to a non-JADE environment within a Windows operating system, you must specify the name of the Web application to which users can connect as a section name within the **jadehttp.ini** file, as follows.

[*application-name*]

An attempt to attach to an application is made only to the applications whose names are specified as section names in the **jadehttp.ini** file.

For details about the paths that the **jadehttp** library file derives for files, see [[Jadehttp Files](#)] Section" under "[Configuring JadeHttp for Remote Connections](#)", in Chapter 2 of the *JADE Installation and Configuration Guide*. See also "[Configuring JadeHttp for Remote Connections](#)" and "[Controlling the Location of Files Uploaded via a Web Application](#)" under "[Configuring Your JADE Software](#)", in Chapter 2 of the *JADE Installation and Configuration Guide*.

Tips If you want to restrict the number of Web-enabled, Web services, or REST services applications, you can set the **LimitPortRange** parameter in the [[WebOptions](#)] section of the JADE initialization file to **true**. This prevents applications from using a port number that exceeds the starting port number plus the number of copies of the application. When you set this parameter to **true**, an application cannot be started when the port limit is exceeded, and a message is output to the **jommsg.log** file.

If you want to change the number of seconds after which Web-enabled, Web services, or REST services applications wait for message transfers to complete before timing out, specify the **ReadTimeout** parameter in the [[WebOptions](#)] section of the JADE initialization file with the appropriate number of seconds. The default value is **600** seconds (10 minutes) but you can specify this parameter and set it to zero (**0**) if you do not want the application to time out waiting for message transfers to complete. (You can prefix this parameter with the name of the application if you want different applications to have different message transfer timeout values.)

Creating Web Pages

The New Form dialog in the JADE Painter provides the **Web** form style option, to enable you to define your Web pages.

As HTML does not allow for exact placement of controls nor the exact width and height, forms painted for the Internet are not "what you see is what you get" (WYSIWYG). Your HTML-generated page may therefore look different from your painted form. Experimentation and experience will enable you to design your forms to achieve the best results.

The **paint** event is not called when running a Web-enabled JADE application, as no JADE forms are created and displayed. Your Web page may look different from your painted form for the following reasons.

- The JADE Painter allows the exact placement of controls, but HTML does not. HTML tables are used extensively to try to place controls.
- In HTML, font sizes are specified as being between sizes 1 and 7, whereas there is greater control over font sizes in JADE. This requires a mapping of font sizes.

- The actual width of input items (for example, combo boxes) is dependent on the content and cannot be controlled using HTML. Other controls, while giving some level of flexibility, also fall into the same category.

Tip Ultimately, the layout of the Web page is determined by the Web browser. The same form can therefore look different in two different browsers. Even a Web-specific painter cannot always accurately generate HTML to reflect what has been painted. For this reason, keep your Web forms simple and always check the output against the Web browsers in which you expect your application to be used.

To reduce the amount of memory being used by Web sessions, JADE creates a physical window only for **Ocx**, **OleControl**, **ActiveXControl**, and MultiMedia controls and a form only if an **Ocx**, **OleControl**, **ActiveXControl**, or **MultiMedia** control is created. In most cases, the **Window** class **hwnd** method returns a null value unless a physical window or a form is created for an **Ocx**, **OleControl**, **ActiveXControl**, or **MultiMedia** control.

For report-type Web pages, the **writeHTML** method of the **Frame** class functions like the **print** method of the **Printer** class, except that HTML code is generated instead of output being directed to the printer or print preview when a Web session is active. If there is no active Web session, the behavior is that of the **print** method (that is, output is directed to the printer or print preview).

» To create a Web page from the JADE Painter

1. Select the **New Form** command from the File menu.

Alternatively, click the **New Form** toolbar button.





The New Form dialog is then displayed, to enable you to define your Web page.

2. Specify a form name in the **Form Name** text box. You must specify a form name, which cannot be the name of another class in the current schema. The form name can be no longer than 29 characters. It can include numbers and underscore characters, but cannot include punctuation or spaces. The first letter of the name is converted to an uppercase character, if it is lowercase.
3. If your form is a subform, perform one of the following actions.
 - Specify its superform in the **Sub-Form of** combo box.
 - Select its superform from the **Sub-Form of** combo box.
 - Select its superform from the **Existing Forms** list box.
4. In the Form Style group box, select the **Web** option button. The JADE Painter switches to a mode that supports HTML-style windows. The grid is automatically set to support character mode painting (that is, snap-to-grid, with grid coordinates of 8 by 20 pixels for a 640 by 480 pixel display).
5. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

Adding Controls to a Web Page

Use the **Add Control** command from the JADE Painter Controls menu to add a new control to your current form. The **Add Control Type** combo box is then displayed, to enable you to select the type of control that you want to add. (You can also add a Web control by using the appropriate Painter palette button.) For details, see "[Adding Controls to Your Form](#)", in Chapter 5 of the *JADE Development Environment User's Guide*.

In addition to the Painter controls listed under "[Generating HTML](#)", later in this chapter, the Web controls listed in the following table are also provided. (For details, see [Chapter 2](#) of the *JADE Encyclopaedia of Classes*.)

Control	Palette Button	Description
WebHotSpot		Inserts a rectangular hotspot directly on to a picture control
WebHTML		Inserts text directly into the generated HTML
WebInsert		Inserts the contents of a file as part of the HTML generation
WebJavaApplet		Inserts a Java applet into the generated HTML

The following controls are *not* supported on Web pages, and are ignored in the HTML generation process.

- Base controls, subclassed to create your own controls
- Browse buttons
- Horizontal and vertical scroll bars
- OLE control
- Progress bar
- User-defined subclassed controls

Tip Use the [Table](#) class [borderStyle](#) property setting of **BorderStyle_None** if you want to turn off the display of borders around tables on your Web pages.

Generating HTML

The generation of your HTML code starts when the [show](#) method of a form is called or the start-up form is initiated, as follows.

1. The [load](#) method of the form is invoked. This method must set up all required information in the controls; for example, a combo box must be populated.
2. When the load process has completed, the HTML generation is started. This generation creates a string containing HTML text.
3. The string of generated HTML text is then returned to the Web browser, which displays the information.

See also "[Adding and Maintaining HTML Documents](#)", in Chapter 12 of the *JADE Development Environment User's Guide*.

Enhanced Web Functionality

The [Application](#), [WebSession](#), [Form](#), and [Window](#) classes provide methods and properties that enable you to enhance the handling of your JADE applications deployed on a Web browser.

For details, see Chapters 1 and 2 of the *JADE Encyclopaedia of Classes*. For a summary, see the following subsections.

Application::webMinimumResponseTime Property

The **Application** class **webMinimumResponseTime** property is summarized in the following table.

Property	Description
webMinimumResponseTime	Contains the maximum time (in seconds) that a Web browser user waits before a response must be sent back to that browser user

This property can be set dynamically at run time or you can set it in the JADE development environment by using the **Minimum Responses Time** text box in the Web Options sheet of the Define Application dialog.

Application Class Methods

The **Application** class provides methods that you can reimplement in your applications; for example, so that messages are displayed on Web browsers to advise users of Web session events. These methods are summarized in the following table.

Method	Description
executeMethodNotFoundMessage	Returns a default HTML string to a Web browser user when the method specified for execution cannot be found or it is invalid
createSessionErrorMessage	Returns the message displayed on Web browsers when a Web session cannot be created
getCurrentSession	Returns a reference to the WebSession object of the specified Web session identifier
getCurrentSessionId	Returns a string of up to 16 characters that identifies the current Web session
getSessionTimeout	Returns the Web session timeout value specified for the application
htmlPageNotFoundMessage	Returns the error message that is sent to the receiver when the requested page for the Web application is not found
invalidWebSessionMessage	Returns an HTML string that is to be displayed on the Web browser when a session is invalid
licencesExceededMessage	Returns the message that is displayed on Web browsers when your licences have been exceeded
minimumResponseTimeExceededMsg	Returns a default HTML string to a Web browser user when the maximum wait time or a response is exceeded
removeSessionMessage	Returns the message that is displayed on Web browsers when your Web session ends
setSessionTimeout	Dynamically sets the period in minutes at which the Web session ends if no requests have been received within that time
setStatusLineDisplay	Dynamically changes scrolling text displayed in the Web browser status line
timedOutSessionMessage	Returns the message that is displayed on Web browsers when your Web session times out

Form Class Methods

The **Form** class **allowWebPrinting**, **generateHTML** and **generateHTMLStatic** methods are summarized in the following table.

Method	Description
allowWebPrinting	<p>Enables you to set the allow parameter to true so that Microsoft Internet Explorer 4 and higher Web browsers generate slightly different code. (This parameter is set to false, by default.)</p> <p>Setting the allow parameter to true for these Web browsers enables the correct printing of the contents of the Web page when the Print command is selected in the Web browser File menu. The requirement of this setting is content-dependent (for example, sometimes Internet Explorer 5.5 or higher may not print segments or pages of the displayed Web page because of the way that Internet Explorer handles the style sheet settings).</p>
generateHTML	<p>Generates the HTML string for the form, which can be subsequently manipulated before being sent back to the Web browser. It can also be saved as a file to obtain a snapshot and this file can then be periodically updated (using notifications, for example). This helps reduce generation and Graphics Device Interface (GDI) overhead in situations where the data seldom changes or it is not necessary to have the most current data available.</p> <p>When the generateHTML method is called to generate an HTML string, the HTML is generated without word wrapping when the wordWrap property is set to the default value of false. Set this property to true if you want an HTML string in a table cell generated with word wrapping.</p>
generateHTMLStatic	<p>Generates the static HTML string for the form and builds the full Uniform Resource Locator (URL) action line using the specified Web server machine name, virtual directory on the Web server, and protocol for transmitting data (for example, HTTP or HTTPS).</p>

WebSession Class Properties

The **WebSession** class properties are summarized in the following table.

Property	Contains the ...
lastAccessTime	Timestamp of the last access of the JADE schema in the session.
sessionId	Unique random number identifier of the session.
startTime	Timestamp of the time that the session was started.

WebSession Class Methods

The **WebSession** class provides the methods summarized in the following table.

Method	Description
browserType	Returns a string containing the type of Web browser.
createVirtualDirectoryFile	Passes image files created by a JADE application to the JadeHttp library or JADE mod_jadehttp module.
deleteVirtualDirectoryFile	Deletes specified files from the virtual directory used by the jadehttp library.

Method	Description
getHttpParam	Returns the value associated with the specified HyperText Transfer Protocol (HTTP) parameter.
getHttpString	Returns the HTTP string that is returned from the Web browser.
getServerVariable	Returns the HyperText Transfer Protocol (HTTP) header information for your Web request from the Web server (that is, Microsoft Internet Information Server (IIS) or Apache HTTP Server).
getSessionForm	Keeps track of all open form instances in the current session in a Web-enabled application that uses JADE forms.
getWebSessionCount	Returns the total number of active Web sessions for all nodes connected to the JADE server.
isVDFilePresent	Returns whether the requested file is present on the Web server side of the firewall when using the JADE Web interface via the jadehttp library file or the JADE mod_jadehttp module.
processRequest	Executed when a request is received from the Web. The appropriate form is then updated with the information received from the incoming string and a reply is sent back to the Web browser after all processing is complete.
removeSession	Removes the current Web session.
removeSessionWithMessage	Enables you to remove the current Web session and send the specified message.
reply	Executed when all processing is complete and the JADE system is ready to send a response back to the Web browser.
setCurrentLocale	Dynamically sets the current locale to the specified value.

Window::userScript Property

The **Window** class **userScript** property is summarized in the following table.

Property	Description
userScript	String containing set up scripts that are to be included as part of the HTML generation.

Window Class Web-Related Methods

The **Window** class Web-related methods are summarized in the following table.

Method	Description
addWebEventMapping	Adds functions to be invoked when a specified event occurs
clearWebEventMappings	Removes all Web event mappings for a specified window
getWebEventMappings	Returns all of the Web event mappings for a specified window
removeWebEventMapping	Removes the specified event for window

Generating a Web Image

When an image, or picture, is encountered on a Web page, the image file is created as part of the HTML generation at run time, and stored in Graphics Interchange Format (GIF), Joint Photographic Experts Group (JPEG), or Portable Network Graphics (PNG) image format for use with Web browsers.

Handling Web Forms in Your JADE Code

If your JADE application is HTML-enabled and it is invoked as a Web application, the display of your Web page is determined by the methods of the **Form** class listed in the following table.

Method	Action
show	Executes the load method, generates the HTML, and returns the HTML to the Web server.
showModal	Not supported, and generates an error at run time or actions the show method. (For details, see " Enabling Your JADE Application for HTML Thin Client Access ", later in this chapter.)
unloadForm	Removes the form from the Web session list of open forms.

In addition, you can use the **Form** class:

- [webBrowserAutoRefreshInterval](#) property, to specify the number of seconds after which the Web page is automatically refreshed and when that property is set to a non-zero value.

The default value of zero (**0**) for the automatic refresh interval indicates that the Web page does not refresh automatically. If you specify an automatic refresh interval and you do not specify the URL to invoke when the number of seconds is reached, control is returned to the JADE application.
- [webBrowserAutoRefreshURL](#) property, to specify the URL to invoke when the automatic refresh interval is reached.
- [webBrowserDisableBackButton](#) property, to specify whether the currently displayed Web page refreshed instead of displaying the previous page when the user clicks the Web browser **Back** button. As it is not possible to disable the **Back** button or the menu item from scripting, set this property to **true** if you want to prevent the user from going to a previous Web page.

Handling Events on Web Pages

The [click](#) events for controls on your Web page are executed if they are present in your JADE code, with the exceptions that instead of the **click** event, the [sheetChg](#) event is executed for a folder control on a Web page if it is present in your JADE code.

If a **click** event is present, the generated HTML sets up the Uniform Resource Locator (URL) to cause a **click** event on the control to send a request back to the JADE application.

No other event is processed, unless it is explicitly called from the **click** event method.

Notes Because of HTML restrictions, the **click** event cannot be executed for text box controls. Excessive use of the **click** event significantly slows HTML thin client interaction, as each event is passed back to the JADE application for processing.

The **paint** event is not called when running a Web-enabled JADE application, as no JADE forms are created and displayed.

Events are supported only on Web pages accessed by using Internet Explorer 4.0 (or higher).

The transient **RootSchemaSession** class provides the protected **allowHiddenControlEvents** property, which specifies whether hidden controls on Web pages can invoke event methods. As this property is **false** by default, set it to **true** to specify that event methods can be invoked by hidden controls.

For details about handling supported event methods in controls on Web pages, see "[Enabling Event Handling on Web Pages](#)", later in this chapter.

When a request is sent to the JADE application, the Web browser waits for a reply. If the JADE code that is processed does not initiate the creation of another Web page, the form that was previously displayed and not unloaded is displayed again. If there were no previous forms, the start-up form is displayed so that the Web browser does not wait indefinitely for a response.

Multiple Selections in List Boxes on Web Forms

You must use the Ctrl key or Shift key to select multiple items in a list box on a Web form, as HTML does not support the **MultiSelect_Simple** functionality. Multiple selections are therefore regarded as extended multiple selections (that is, **MultiSelect_Extended**, where the Shift+click or Shift+arrow key extends the selection from the previously selected item to the current item or Ctrl+click selects or deselects an item in the list).

Enabling Your JADE Application for HTML Thin Client Access

You can enable your JADE application for HTML thin client access when you first define your application or you can access the Define Application dialog at any time to enable HTML thin client access or to maintain your application preferences.

For more details about the Define Application dialog, see "[Defining Applications](#)", in Chapter 3 of the *JADE Development Environment User's Guide*.

» To access the Web Options sheet of the Define Application dialog

1. Perform one of the following actions to open an Application Browser window.
 - Click the **Browse Applications** button from the browse toolbar
 - Select the **Applications** command from the Browse menu, or press Ctrl+L
2. From the Application menu of the Application Browser, select the **Add** command to add a new application to your current schema or the **Change** command to enable HTML thin client access or to maintain the preferences of your current application.

The **Application** sheet of the Define Application dialog is then displayed.

3. Select the **Web-Enabled** or **Web-Enabled Non-GUI** option in the **Application Type** combo box, to specify that the application can be accessed from the Web using an HTML thin client. The **Web Options** sheet is then enabled.

The start-up form defined for the application is the first Web page that is displayed when the application is invoked from a Web browser. Application features such as Multiple Document Interface (MDI) forms and three-dimensional controls are ignored for HTML-enabled applications.

In addition, as applications of type **Web-Enabled Non-GUI** do not display the Web Application Monitor window, this type of application can be run in the background.

4. Click the tab of the **Web Options** sheet.

The **Web Options** sheet is then displayed. Now that you have enabled your application for HTML thin clients, you can specify your access options.

Tip Use the [currentSession](#) system variable in a JADE method to determine if a transaction is from another workstation or from an HTML thin client. The **currentSession** system variable is set to **null** if there is no current Web session.

For details, see "[System Variables](#)", in Chapter 1 of the *JADE Developer's Reference*. See also Chapter 12 of the *JADE Development Environment User's Guide*, "[Adding and Maintaining HTML Documents](#)".

Specifying Your HTML Thin Client Access Options

An example of the **Web Options** sheet of the Define Application dialog is shown in the following image.

The screenshot shows the 'Define Application' dialog box with the 'Web Options' tab selected. The dialog has three main tabs: 'Application', 'Form', and 'Web Options'. Under 'Web Options', there are sub-tabs for 'HTML Documents', 'Web Services', and 'JADE Forms'. The 'HTML Documents' sub-tab is active, showing fields for 'Image Directory Mapping' (Virtual Directory: /images, Physical Directory: C:\Temp), 'Maximum HTML Size' (0 K), and 'Scrolling Text' (Welcome to Erewhon Investments Inc.). There are also checkboxes for 'Disable Messages', 'Treat showModal as show', 'Cross Browser Compatibility', and 'Form Style Display (IE 4)'. A 'Web Events...' button is also present. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

For more details about the Define Application dialog, see "[Defining Applications](#)", in Chapter 3 of the *JADE Development Environment User's Guide*. For details about defining a Web services application, see "[Defining a Web Services Provider Application](#)", in Chapter 11 of the *JADE Developer's Reference*. For details about defining a REST services application, see "[Defining a REST Service Application](#)", in Chapter 11 of the *JADE Developer's Reference*.

» To specify your HTML thin client options

1. In the **Connection Name** text box, specify the appropriate TCP/IP address in the format *TCP-address:port-number* (for example, **143.67.78.90:6014**).

Tip If you want to get the IP address, call the **WebSession** class **getServerVariable** method to return the IP address of the current Web session as determined by the Web server, as follows.

```
currentSession.getServerVariable('REMOTE_ADDR');
```

For details about using the **ConnectionName** parameter in the [WebOptions] section of the JADE initialization file, see your *JADE Initialization File Reference*.

Note The connection name defaults to the application name. (The application name is displayed in the title bar of the Web Application Monitor window.) If you have more than one application with the same name, specify a unique name to identify the connection. The connection name can include any character or number in the ranges **A** through **Z**, **a** through **z**, or **0** through **9**. The first character must be uppercase.

2. In the **Application Copies** text box, specify the maximum number of copies that you require for your HTML thin client application. When the HTML thin client application starts, the number of copies specified in this text box are started. The default value is **1**. You can specify a maximum of 62 copies, which is a Windows-imposed limit.

Note All application copies use the same port number. If you specify **6014** as the starting port number in the **Connection Name** text box and specify three in the **Application Copies** text box, the application copies use port numbers **6014**, **6015**, and **6016**.

For details about using the **ApplicationCopies** parameter in the [WebOptions] section of the JADE initialization file to specify or change the number of copies of the application to start up, see your *JADE Initialization File Reference*.

3. In the **Session Timeout** text box, specify the period in minutes at which the Web session is to end if no requests have been received within that time. The default value of zero (**0**) indicates infinity; that is, you do not want your Web session to time out.

The **Application** class **getSessionTimeout** and **setSessionTimeout** methods enable you to get and dynamically set the current Web session timeout value, respectively.

4. In the **Minimum Response Time** text box, specify the maximum time (in seconds) that a Web browser user has to wait before a response must be sent back to that user from the JADE application, triggered by a timer event.

By default, requests do not time out; that is, the default value of zero (**0**) indicates infinity.

When the timer event occurs, a default message is sent back to the browser and the current request is terminated. You can override the default message that is sent to the browser user by reimplementing the **Application** class **minimumResponseTimeExceededMsg** method in the **Application** subclass of your user-defined schema.

You can also set the minimum response time value dynamically at run time, by setting the **Application** subclass **webMinimumResponseTime** property in your logic.

5. Check the **Disable Messages** check box to specify that your Web session messages are not displayed in a message box on your client workstation; for example, in a confirmation dialog when the Web session is shut down.

By default, message boxes are enabled; that is, this check box is unchecked. Regardless of the setting of this parameter, any message that is raised is displayed in the Web Application Monitor window.

6. If you selected the **HTML Documents** option button on the **Application** sheet of the Define Application dialog (that is, you want HTML generated based on HTML documents in your Web application), the **HTML Documents** sheet on the **Web Options** sheet is enabled. (For details about the **Applications** sheet, see

"[Defining Applications](#)", in Chapter 3 of your *JADE Development Environment User's Guide*.)

Perform the following actions to define your HTML document requirements.

- a. In the **Machine Name** text box, specify the physical location to be used when generating the HTML for the **JadeHTMLClass** class **buildFormActionOnly** and **buildLink** methods.
- b. In the **Virtual Directory** text box, specify the Uniform Resource Location (URL) to be used when generating the HTML for the **JadeHTMLClass** class **buildFormActionOnly** and **buildLink** methods.

For details about using the **URLSpecifications** parameter in the [\[WebOptions\]](#) section of the JADE initialization file to specify Internet server virtual directories for HTML documents for all of your Web-enabled applications or for a specific Web-enabled application, see the *JADE Initialization File Reference*.

- c. Use the **Home Page** combo box to select the HTML home page that is to be the start-up page when you run your Web-enabled application. (If you do not select a home page, an error is displayed when you click the **OK** button on the Define Application dialog.)

See also "[Adding and Maintaining HTML Documents](#)", in Chapter 12 of the *JADE Development Environment User's Guide*.

7. If your Web application uses JADE forms from which you want HTML dynamically generated and you did not change the default value of **JADE Forms** in the Web Application Type group box on the **Application** sheet of the Define Application dialog, specify your Web application form options on the **JADE Forms** sheet.

Note The **JADE Forms** sheet is enabled only when your application does *not* use HTML documents; that is, you selected the form that is to be displayed when the new application is started in the **Start-up Form** combo box on the **Application** sheet of the Define Application dialog and you did not change the default Web application type of **JADE Forms**.

Perform the following actions to define your JADE form requirements.

- a. In the **Virtual Directory** text box, specify the Uniform Resource Location (URL) if you want your images and Java applets located in a directory other than the working directory of the JADE application. There is no default base URL. If you do not specify a URL in this text box, the virtual directory specified in the Microsoft IIS is used as the URL.

When setting up IIS for JADE, you must specify a virtual directory and the physical location of this directory. By default, the virtual directory is used as a work area for generating image files.

If the Base URL is set up, the generated HTML code uses this URL to set up locations for these files, rather than the default work area.

To enable JADE to determine the physical location of this base URL, you must also specify the physical directory in the **Physical Directory** text box. To create the physical directory, click the **Create** button. In IIS, this physical directory must be set up as a virtual directory, by using the base URL that you specify in this text box.

The **VirtualDirectory** parameter in the JADE initialization file [\[WebOptions\]](#) section enables you to specify a location for your images and Java applets other than the working directory specified in this text box, if required. (For details, see "Web Options Section [\[WebOptions\]](#)", in the *JADE Initialization File Reference*.)

For details about using the **URLSpecifications** parameter in the [\[WebOptions\]](#) section of the JADE initialization file to specify Internet server virtual directories for HTML documents for all of your Web-enabled applications or for a specific Web-enabled application, see the *JADE Initialization File Reference*.

- b. In the **Physical Directory** text box, specify the name of the physical working directory for your

HTML-enabled application; for example, `s:\jade\webtest\bin`. The working-directory is usually the directory in which the JADE executable file (`jade.exe`) is located but you can specify another working directory when you install JADE, if required. This is used by JADE to generate images for the Web.

The **PhysicalDirectory** parameter in the JADE initialization file [WebOptions] section enables you to specify a physical directory to override the working directory specified in this text box, if required. You can specify multiple physical directories in the same JADE initialization file. For details, see "Web Options Section [WebOptions]", in the *JADE Initialization File Reference*.

Note This parameter, which is used to generate images for the Web, applies only to Web-enabled (that is, HTML-enabled) applications and not to Web services or REST services applications.

- c. In the **Maximum HTML Size** text box, specify the maximum size in K bytes of the HTML string that can be generated before stopping the generation. Although you can specify 99,999K bytes as the size of the maximum HTML string, this would have a considerable impact on performance.

The optimum HTML string size is dependent on the available memory of both your Internet and your JADE server. 30K bytes is a recommended maximum string size.

If the specified maximum HTML string is generated, the HTML generation is then stopped and the partially generated string is returned to the Internet browser with the following message appended to the string.

```
Exceeded maximum lines
```

The default value of zero (**0**) specifies that there is no maximum HTML size.

- d. In the **Scrolling Text** text box, specify the text that you want to scroll across the bottom of your Web page, if required. By default, no text is scrolled across a Web page.
- e. Check the **Treat showModal as show** check box to display Web pages in the application by using the `show` method if the `showModal` method is encountered in your code. (The `show` method executes the `load` method, generates the HTML, and then returns the HTML to the Web server.)

As the `showModal` method is not supported, by default an error is raised at run time if the `showModal` method is encountered.
- f. Check the **Cross Browser Compatibility** check box to specify the version of HTML code that JADE generates for your Web sessions. Check this box if you want compatibility across browsers for your JADE Web sessions; that is, identical HTML code is generated for both Netscape and Internet Explorer.

By default, different HTML code is generated for Internet Explorer 4.0 (that is, this check box is unchecked).
- g. Check the **Form Style Display (IE 4)** check box to specify that your Web pages are displayed as windows (that is, with captions and borders) when using Internet Explorer 4.0. By default, your JADE forms are displayed as Web browser pages (that is, this check box is unchecked).
- h. Click the **Web Events** button on the **JADE Forms** sheet if you want to specify the handling of:
 - The `click` event method in check box, combo box, list box, table, and option button controls on Web pages, and in your user-defined subclasses of those five controls types only.
 - The `sheetChg` event method in folder controls and your user-defined folder control subclasses.

Note Events are supported on Web pages only when using Microsoft Internet Explorer 4.0 or higher.

The Web Events dialog is then displayed. For details, see "Enabling Event Handling on Web Pages", later in this chapter.

8. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

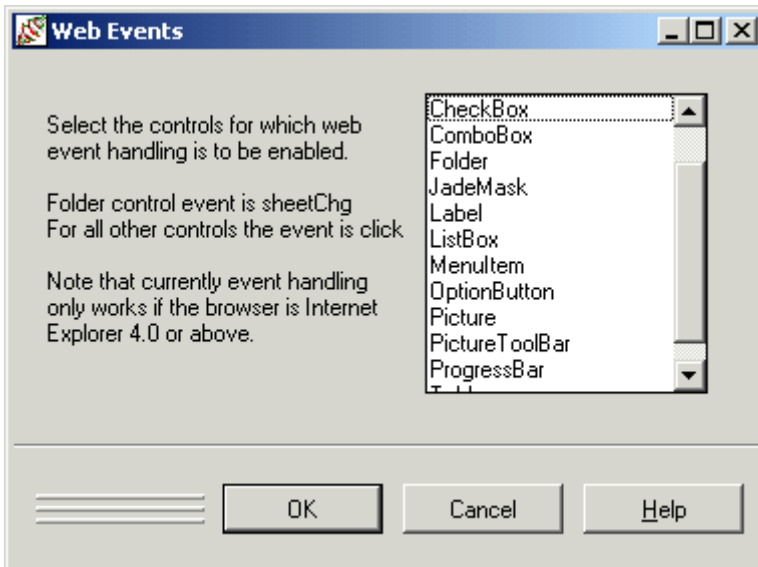
For details about implementing a TCP/IP connection to your JADE applications, see "[Connecting to JADE Applications from Internet Information Server \(IIS\)](#)", in Chapter 2 of the *JADE Installation and Configuration Guide*.

Tips If you want to restrict the number of Web-enabled, Web services, or REST services applications, you can set the [LimitPortRange](#) parameter in the [\[WebOptions\]](#) section of the JADE initialization file to **true**. This prevents applications from using a port number that exceeds the starting port number plus the number of copies of the application. When you set this parameter to **true**, an application cannot be started when the port limit is exceeded, and a message is output to the **jommsg.log** file.

If you want to change the number of seconds after which Web-enabled, Web services, or REST services applications wait for message transfers to complete before timing out, specify the [ReadTimeout](#) parameter in the [\[WebOptions\]](#) section of the JADE initialization file with the appropriate number of seconds. The default value is **600** seconds (10 minutes) but you can specify this parameter and set it to zero (**0**) if you do not want the application to time out waiting for message transfers to complete. (You can prefix this parameter with the name of the application if you want different applications to have different message transfer timeout values.)

Enabling Event Handling on Web Pages

The Web Events dialog, shown in the following image, is displayed when you click the **Web Events** button in the Define Application dialog **Web Options** sheet.



When you are using Internet Explorer 4.0 (or higher) for your HTML thin client applications, you can enable:

- The **click** event for the list box, combo box, option button, table, and check box controls or your user-defined subclasses of those five control types only.

The conditions that must be satisfied to enable the **click** event for controls displayed on Web pages are as follows.

- The subclassed control must be selected in the Web Events dialog.
- Browser compatibility mode must be turned off, by ensuring that the **Cross Browser Compatibility**

check box on the **Web Options** sheet of the Define Application dialog is unchecked.

- There must be a **click** event for the control on the form.
- The **sheetChg** event method in folder controls and your user-defined folder control subclasses.

» To enable support for events on your Web pages

1. In the list box, select the control or controls for which the applicable **click** or **sheetChg** event is to be enabled on Web pages. (By default, these events are not supported on the controls on Web pages.)

If you have subclassed any of these controls in your JADE application, those subclasses are also listed to enable you to support the appropriate event on your user-defined controls.

Tip Use the Ctrl key to make multiple selections, if required.

2. Click the **OK** button. Alternatively, click the **Cancel** button to abandon your selections.

Using a Web Browser to Access Your JADE Application

The HTML code that is generated by JADE conforms to the HTML 3.2 Specification. Specific Netscape or Explorer extensions are not used. You can add these extensions (for example, ActiveX controls) by inserting raw HTML into a Web page, if required.

For details about generating HTML code that is compatible across browsers for your JADE Web sessions (that is, identical HTML code is generated for both Netscape and Internet Explorer), see "[Specifying Your HTML Thin Client Access Options](#)", earlier in this chapter. See also "[Form Class Methods](#)" under "[Enhanced Web Functionality](#)", earlier in this chapter, for details about the [allowWebPrinting](#) method.

Accessing a JADE Application from an HTML Thin Client

To access JADE from an HTML thin client, set up a virtual directory in your Internet Service Manager, as appropriate. For details about invoking a user method, see the following subsection.

Invoking a User Method

To allow a Web session to bypass the standard processing loop, you can set up a command in the **http** string returned from the Web browser. This command has the following format.

```
_executeMethod=class-name::web_method-name
```

Note When you invoke a user method that bypasses the standard processing loop, your method name must consist of the **web_** name prefix.

For example, if the **http** string is like that shown in the following example, the **web_generateXML** method in the **Transaction** class will be executed, passing the **http** string as a parameter.

```
http://persephone/jade/jadehttp.dll?MyWebApp&_executeMethod=Transaction::web_generateXML&TransactionType=CashOnly
```

The method that is executed must return a primitive type, which is converted to a **Binary** primitive type before being sent back to the Web browser. It is expected that the method will return a **String** value. In a Unicode JADE system, the contents of the returned string are converted from Unicode to UTF-8 before being sent back to the Web browser.

If the method does not exist or the return type is invalid, a default message is sent back to the Web browser. To override this message, you can reimplement the [executeMethodNotFoundMessage](#) method in the [Application](#) class of a user-defined schema.

JADE Connectivity from the Internet

When using a Web service consumer or the [JadeHTTPConnection](#) class, the default communications protocol is the WinHTTP library. WinHTTP is more appropriate for server-type environments; WinINET is more appropriate for low-performance client situations.

The processing of requests from the Web server is handled as follows.

- The **jadehttp** library file, supplied with JADE on the release medium and located on your Internet server, is directly called by the Web server for each HTML client request that is made. This library connects to a JADE client node by a named pipe or a TCP/IP connection.

The **jadehttp** library file sends the received request over the channel to the JADE application, which processes the request and then returns an HTML page for transmission to the HTML thin client user.

- The [InternetPipe](#) or [JadeInternetTCPIPConnection](#) class enables JADE to establish a communications channel with another program. To communicate with the jadehttp library file, the JADE application creates an instance of the appropriate class, and then offers the named pipe or TCP/IP for opening.

When the library opens the other end of the channel, JADE waits for input from the library. When input arrives, JADE processes that input and then sends the reply back to the library. JADE then waits for more input.

As JADE asynchronously waits for input from the named pipe or TCP/IP, the JADE client node can perform other tasks when it is idle; for example, monitoring and displaying the system processing status.

Connections from the **jadehttp** module and JADE Web application can be over TCP/IP or a named pipe connection. JADE systems use the named pipe connection by default. For details about specifying that the HTML generation for Netscape and Mozilla browsers is similar to that for Internet Explorer, see the [UseHTML4ForNetscape](#) parameter in the [\[WebOptions\]](#) section of the JADE initialization file, in the *JADE Initialization File Reference*.

Note The main advantage of a TCP/IP connection over a named pipe connection is that the machine hosting IIS or Apache HTTP Server can be different from the machine that is running the JADE application, to provide greater security via firewalls. A TCP/IP connection is also slightly faster than a named pipe connection and an Apache HTTP Server cannot connect over a named pipe.

For details, see "[Connecting to JADE Applications from Internet Information Server \(IIS\)](#)", in Chapter 2 of the *JADE Installation and Configuration Guide*.

WinHTTP and WinINET are operating system libraries that perform lower-level network communications. WinINET proxy settings are set the same way as proxy settings for Internet Explorer (that is, by using the Internet Options dialog accessed from the Control Panel).

For details about setting Internet proxy settings, see Microsoft documentation. WinHTTP proxy settings are configured using the **netsh.exe** tool. To obtain help, specify the following command.

```
netsh winhttp set proxy help
```

See also the [EnableWinHTTP](#) and [EnableWinINET](#) parameters in the [\[JadeEnvironment\]](#) section of the JADE initialization file. If you set both parameters to **true**, WinHTTP is used in preference. If you set both parameters to **false**, the support of both protocols is disabled and a Web service consumer can use only the JADE Direct scheme; that is, **jadehttp.tcp**.

Managing Your Web Sessions

When an HTML client request is received by your JADE application, the Web session manager determines whether the session is for an existing session. If there is no existing session, a new instance of **WebSession** subclass is created. (For details, see "[WebSession Class](#)", in [Chapter 1](#) of the *JADE Encyclopaedia of Classes*.)

The last access timestamp determines the disconnect status. If there is no activity for the session for the period of time specified in the **Session Timeout** text box of the Define Application dialog, the session is terminated. If you transmit a request after the specified timeout period, you are informed that the Web session has timed out, you must reconnect to the JADE application, and then sign on again.

You can reimplement the **Application::createSessionErrorMessage** method if you want to display a different session error message on the Web browser when a Web session cannot be created. The returned string should be in HTML format, for correct rendering on the browser.

The **Application** class provides the **getCurrentSession**, **getCurrentSessionId**, **getSessionTimeout**, and **setSessionTimeout** methods, to enable you to return the current Web session, its identifier, the Web session timeout value specified for the application, and dynamically set the timeout period for all Web sessions that are subsequently created, respectively. For details, see [Chapter 1](#) of the *JADE Encyclopaedia of Classes*.

Initiation and Process Flow

You can make each schema in the JADE database HTML thin client-capable, by defining an HTML-enabled application instance. When it is initiated, this JADE application then communicates with the **jadehttp** library that is located on the Internet server.

By default, this application opens an instance of the **NamedPipe** or **JadeInternetTCPConnection** class with the name of the JADE application as its name, and waits for the **jadehttp** library to connect to the other end of the pipe or TCP/IP. When the pipe or TCP/IP is connected, it waits for HTML client requests to be sent over the pipe or TCP/IP. For details about implementing a TCP/IP connection, see "[Connecting to JADE Applications from Internet Information Server \(IIS\)](#)", in [Chapter 2](#) of the *JADE Installation and Configuration Guide*.

When the first request for the JADE application is received, the IIS initiates the **jadehttp** library, and then calls the **GetExtensionVersion** entry point in the library to obtain the Internet Server Application Programming Interface (ISAPI) version that is being used. The library is initialized as part of this call, and it attempts to open a pipe or TCP/IP connection by using the JADE server node name.

Note To process multiple HTML client requests simultaneously, run additional copies of the JADE application. For details, see "[Handling Multiple Copies of the JADE Program](#)", in [Chapter 1](#) of the *JADE Installation and Configuration Guide*. Each copy of the JADE application opens its own instance of the named pipe. The **jadehttp** library uses the additional pipe channels, as required. (By default, there are **10** pipe channels.)

If the connected JADE application terminates and breaks a pipe or TCP/IP channel, the library removes that pipe or TCP/IP from the list of available channels. When the JADE application is restarted, the library reestablishes the channel without any required intervention.

Web sessions are not closed when an application is terminated. They are closed only when the **WebSession** class **removeSession** method is called, a session timeout specified in the **Session Timeout** text box on the **Web Options** sheet of the Define Application dialog occurs (for details, see "[Specifying Your HTML Thin Client Access Options](#)", earlier in this chapter), or when all of the Web-enabled applications terminate normally.

JADE Processing of an HTML Client Request

When the JADE system receives an HTML client request, the following actions are performed.

1. The TCP/IP address and the contents of a hidden field containing the encrypted session identifier are obtained from the input data.
2. The TCP/IP address and the encrypted session identifier are used to search the session dictionary for an existing session object for the client. If the entry is not found in the dictionary, a new client session object is created.
3. A `processRequest` method of the `WebSession` class instance is then called, passing the request data to JADE. This method processes the request, and sends a formatted HTML page back by the named pipe or TCP/IP instance on which the input data was received.

The `jadehttp` library sends this page to the client, and the Web server is then informed that the request processing is complete.

If you want to change the number of seconds after which Web-enabled applications wait for message transfers to complete before timing out, specify the `ReadTimeout` parameter in the `[WebOptions]` section of the JADE initialization file with the appropriate number of seconds. The default value is **600** seconds (10 minutes) but you can specify this parameter and set it to zero (**0**) if you do not want the Web-enabled application to time out waiting for message transfers to complete. (You can prefix this parameter with the name of the application if you want different applications to have different message transfer timeout values.)

You can use the `MaxMessageSize` parameter in the `[application-name]` section of the `jadehttp.ini` file to specify the maximum size allowed for a single-part Web message (that is, a single stream of data with minimal formatting) if you want a value greater than 1,000,000 bytes. (For details, see "[Configuring JadeHttp for Remote Connections](#)", in Chapter 2 of the *JADE Installation and Configuration Guide*.)

When you specify a maximum number of queued entries, any additional request received that would also be queued when the maximum queued entries is reached will be rejected. The user will be sent the contents of a file named `busy.htm` from the same directory as the `jadehttp.ini` file (as it is for the `jadehttp.htm` error file, described in the following section).

If that file is not available, the following text response will be sent.

```
The application is too busy
```

```
This request cannot be processed at this time due to heavy usage - please try again shortly.
```

You can use the `MinMessageSize` parameter in the `[application-name]` section of the `jadehttp.ini` file or the `MinMessageSize` directive in the Apache `mod_jadehttp.so` library to specify the minimum size allowed for a Web message received from JADE using the `WebSession` class `reply` method to send HTML string Web requests back to the client node.

The minimum value is **1** byte, the maximum value **1024** bytes, and the default value **10** bytes. This value is read the first time the specified application is accessed after `jadehttp.dll` has been loaded by Internet Information Server (IIS) or it is read once, when the Apache Web server starts.

Message Logging

If communications cannot be established with the JADE system, a `jadehttp.htm` file is output, advising you that the service is down. This file must be in the same directory as the `jadehttp` library file on the Internet server.

The `jadehttp` library file reads an initialization file called `jadehttp.ini`, located in the same directory as the library.

A new log file is always opened after the **jadehttp.dll** library has been initiated rather than appending to the existing log file. The current date and time is always added to the file name when a new file is created (when the library is initiated or when a log file switch occurs). When a **jadehttp.log** log file reaches 1M byte, a new log file is created.

For details about the parameters available in the [Jadehttp Logging] section of the **jadehttp.ini** file that enable you to specify your message logging requirements during Web-enabled applications, see "[Jadehttp Logging] Section" under "[Configuring JadeHttp for Remote Connections](#)", in Chapter 2 of the *JADE Installation and Configuration Guide*.

Processing problems in the **jadehttp** library are logged to the **jadehttp.log** file, located in the same directory as the library. Failures that occur result from the unavailability of the JADE system.

The IIS server logs all HTML client requests to its own log file. You should enable IIS logging, and set the option to automatically start a new file each day. The **jadehttp** library file also records routine messages to the log when it is initiated and closed, and when connections to the named pipe or TCP/IP channels are made and broken.

Suppressing the Logging of Messages

To ensure the security of data, set the value of the **Trace** parameter in the [Jadehttp Logging] section of the **jadehttp.ini** file for IIS or the **JadeHttp_Trace** directive in the JADE **mod_jadehttp** module for Apache to **true**.

When the value is **true**, messages logged to the **jadehttp.log** file do not include any of the text sent or received from the client, as this text could contain personal information, passwords, credit card details, and so on. Logged messages then only acknowledges that a message has been received or sent, because it is not possible to distinguish what is sensitive data and what is not.

Handling Exceptions in an HTML-Enabled Application

The HTML thin client application is armed with a global exception handler. When an unhandled exception occurs:

1. The error is reported to the HTML thin client user, logged on the JADE client workstation and a message is displayed to inform the JADE client.
2. The operation is aborted.
3. The HTML thin client user is returned to the previous valid form.

You can reimplement the **Application::createSessionErrorMessage** method if you want to display a different session error message on the Web browser when a Web session cannot be created.

Message Box Handling

When an exception occurs in an HTML-enabled JADE application, the **msgBox** method in the **Application** class creates an HTML page and returns to the HTML thin client user.

Caution As there is no modal support with HTML thin client applications, any code following **app.msgBox** in your JADE method continues to be executed.

You should therefore use **app.msgBox** sparingly and with care.

Chapter 2

Monitoring Your Web Sessions

This chapter covers the following topics.

- [Overview](#)
- [Opening a Browser for a Web Services Application](#)
- [Shutting Down the Web-Enabled Application](#)
- [Clearing the Displayed Information](#)
- [Specifying the Content of Logged Messages](#)
- [Disabling Logging](#)
- [Directing Web Monitor Information to a File](#)
- [Restarting a Web Session](#)
- [Closing a Web Session](#)
- [Displaying Session Details](#)
- [Listing Active Web Sessions](#)
- [Displaying Session Statistics](#)
- [Accessing Online Help](#)

Overview

When you run an HTML-enabled application, the JADE Web Application Monitor is automatically initiated on the client node workstation on which the HTML-enabled JADE application is running.

If a client node has more than one active HTML-enabled application, a Web Application Monitor window is displayed for each active application on that workstation.

Use the Web Application Monitor to view the status of Web sessions, open your default browser for Web services applications, and to reset or close Web sessions.

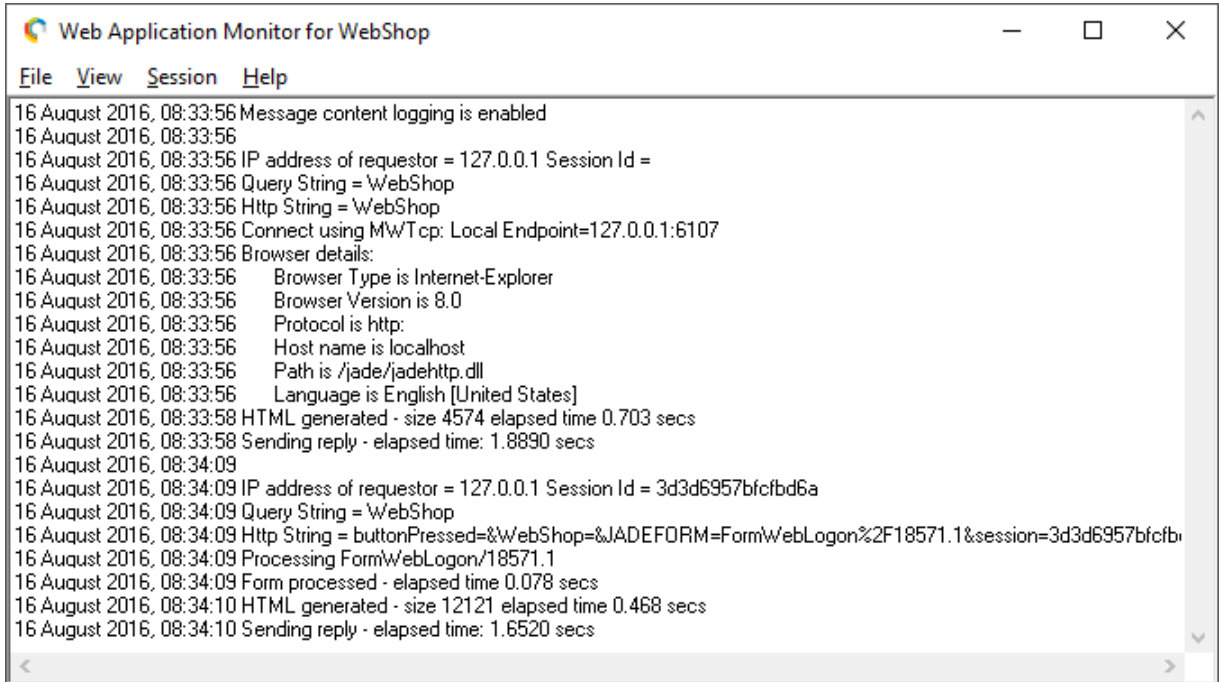
When the Web Application Monitor is initiated, the window displays session sign on information, including:

- Internet protocol (IP) address
- Session id
- Query string

The Web Application Monitor menu bar contains the menus listed in the following table.

Menu	Description
File	Administers your HTML-enabled application
View	Controls the display of information in your Web Application Monitor window
Session	Controls your Web session
Help	Accesses the standard Common User Access (CUA) help options

The Web Application Monitor window is a multiple-line text window that logs and displays Web session information. Information is appended to the window display with each user request and new session interaction, as shown in the following image.



Note You can change the font name and size for the Web Application Monitor window by using the **Editor** sheet of the Preferences dialog, accessed from the **Preferences** command in the Options menu of browse windows in the development environment.

Use the scroll bar to scroll up and down the displayed HTML thin client information, if required.

When you exit from the HTML-enabled application, the Web Application Monitor window is closed, and all displayed information is purged.

For details about running the JADE Monitor that provides system instrumentation and diagnosis, see Chapter 2, "Using the JADE Monitor for System Instrumentation and Diagnosis", in the *JADE Monitor User's Guide*.

Opening a Browser for a Web Services Application

» To open a Web browser and test a Web services application

- Select the **Open Browser** command from the Web Application Monitor File menu.

The following actions then occur.

1. Your default browser (for example, Microsoft Internet Explorer) is started
2. A browser session connects to your Web services application
3. A sample home page for your application is displayed

For details about Web services applications, see Chapter 11, "Building Web Services Applications", in the *JADE Developer's Reference*.

Shutting Down the Web-Enabled Application

Select the **Exit** command from the Web Application Monitor File menu to shut down the HTML-enabled application, clear all related sessions, and close the Web Application Monitor window.

If any users are accessing the application at the time that you select the **Exit** command, a message dialog advises you that there are active users and asks you to confirm that you want to exit from the application.

When you confirm that you want to exit, active users are disconnected when all current transactions have completed and the application is then closed down.

Any user who then tries to access the application again is informed that the application has been shut down.

Clearing the Displayed Information

Select the **Clear Display** command from the Web Application Monitor View menu to clear the information displayed in the text window.

Specifying the Content of Logged Messages

When the JADE Web Application Monitor logging is turned on and the [LogFileName](#) parameter is specified in the [\[WebOptions\]](#) section of the JADE initialization file, the output includes the content of the Web message, which could include personal and privileged information; for example, user codes and passwords.

To control whether JADE logs the content of Web messages, use the [LogMessageContent](#) parameter in the [\[WebOptions\]](#) section of the JADE initialization file. Alternatively, you can specify the XML **logmessagecontent** parameter in the Web application configuration file; that is, the following configuration file parameter disables the logging of message content.

```
<logmessagecontent>false</logmessagecontent>
```

When the **LogMessageContent** parameter in the [\[WebOptions\]](#) section of the JADE initialization file or the XML **logmessagecontent** parameter in the Web application configuration file is set to **true**, any Web logging includes the **Query String = content** and **Http String = content** output. When the value of the **LogMessageContent** initialization file parameter or the XML **logmessagecontent** configuration file parameter is **false**, any Web logging does *not* include this message content.

When you initiate the Web Application monitor and tracing is on, a message is displayed indicating the status of the **LogMessageContent** parameter, as follows.

```
Message content logging is enabled|disabled
```

The message is also displayed if you turn logging on or off by toggling the **Disable Logging|Enable Logging** command in the View menu of the Web Application Monitor window.

Note By default, the value of the **DisableLogging** parameter is **false**; that is, logging is enabled.

Disabling Logging

Select the **Disable Logging** command from the Web Application Monitor View menu to stop the display of information in the monitor window. Web monitor information is logged to the window with each user request and with new Web session information and statistics. For details, see ["Displaying Session Details"](#) and ["Displaying Session Statistics"](#), later in this chapter, and ["Monitoring Your Web Sessions"](#), earlier in this chapter.

When you have selected this command to disable the logging of information to the monitor window, the menu command is toggled to the **Enable Logging** command.

Select the **Enable Logging** command to cause the redisplay of logging information.

Note The **DisableLogging** parameter in the [WebOptions] section of the JADE initialization file specifies whether logging is enabled in the JADE Web Application Monitor window. This parameter is set to **false** by default. You can set it to **true** if you want to stop the display of logging information in the monitor window (for example, when you are running an application as a service and the Web Application Monitor window View menu item cannot be used).

Directing Web Monitor Information to a File

If you want to direct the log information displayed on the Web Application Monitor to a file, use the **LogFileName** parameter in the [WebOptions] section of the JADE initialization file to specify the file name and the full path to which the displayed information is directed for later analysis of transaction times, and so on.

If the specified file name cannot be written to or it is not valid (for example, you did not specify an absolute path or the specified path does not exist), a file called **websession.log** is created in the physical directory specified in the **Physical Directory** text box for your JADE forms on the **Web Options** sheet of the Define Application dialog or in the **PhysicalDirectory** parameter in the [WebOptions] section of the JADE initialization file. (For details about defining the physical directory, see "Specifying Your HTML Thin Client Access Options", in Chapter 1.)

Note The physical directory, which is used to generate images for the Web, applies only to Web-enabled (that is, HTML-enabled) applications and not to Web services or REST services applications.

Restarting a Web Session

Select the **Restart** command from the Web Application Monitor Session menu to force users of a Web session to sign back on to a Web session; for example, in situations where a user is getting unexpected errors.

The Session Id dialog is then displayed. Specify the required Web session identifier in the **Enter Session Id** text box, and then click the **OK** button to force users to restart the session.

Tip Use the **List** command from the Session menu to list all current Web sessions in the Web Application Monitor window. You can then view all current Web sessions, to determine the id of the session that you want to restart.

When you have specified a valid session identifier, all session information relating to open forms is cleared; that is, all currently open forms are unloaded.

The next HTML client request forces the user to the start-up form.

Closing a Session

Select the **Close** command from the Web Application Monitor Session menu to close a current Web session. The Session Id dialog is then displayed.

Specify the required Web session identifier in the **Enter Session Id** text box and then click the **OK** button to force the specified session to close down.

Tip Use the **List** command from the Session menu to list all current Web sessions in the Web Application Monitor window. You can then view all current Web sessions, to determine the id of the session that you want to close.

Any further requests using this session identifier cause the user of the Web browser to be notified that the Web session has ended.

Displaying Session Details

Select the **Details** command from the Web Application Monitor Session menu to display the details of a specific current Web session in the monitor window.

The Session Id dialog is then displayed. Specify the required Web session identifier in the **Enter Session Id** text box and then click the **OK** button to display information for that session.

Tip Use the **List** command from the Session menu to list all current Web sessions in the Web Application Monitor window. You can then view all current Web sessions, to determine the id of the session whose information you want to view.

Each HTML-enabled application has its own list of sessions. The following information is displayed for your specified session.

- IP address
- List of open forms
- Time the session was started
- Time of last access to the session

Listing Active Web Sessions

Select the **List** command from the Web Application Monitor Session menu to list the IP address and encrypted session id of each active Web session.

Use the listed information to select the session that you want to specify in the Session Id dialog to display session details, or to restart or close a Web session.

Displaying Session Statistics

Select the **Statistics** command from the Web Application Monitor Session menu to display statistics of the current Web session in the monitor window.

The following information is displayed for the current Web session.

- Total number of requests
- Minimum response time
- Maximum response time
- Average response time
- Transients remaining for any **JadeHTMLClass** subclasses

Accessing Online Help

Select the **Index** command in the Web Application Monitor Help menu to open online help. The *JADE Web Application Guide* is then opened, providing access to the topics available in online help.

» To access the online help, perform one of the following actions

- Select the **Index** command from the Help menu
- Press F1

The JADE online help is then displayed; for example, the **WebApps.pdf** document is displayed in Adobe Reader.

Use the functions available in JADE online help to find the required topics. For details, see "[JADE HTML5 Online Help](#)" or "[JADE Product Information Library in Portable Document Format](#)", in Chapter 2 of the *JADE Development Environment User's Guide*.

Displaying Information about Your HTML Thin Client Application

Select the **About** command from the Web Application Monitor Help menu to access information about the current HTML-enabled application. The About form for your application is then displayed if you have specified an About form for your application (by using the **About Form** combo box in the **Application** sheet of the Define Application dialog).

If you have not specified an **About** form for your application, the JADE default About box is displayed. This dialog displays the following information, and is for display purposes only.

- Application name
- Current JADE release
- Current year

When you use the JADE Painter menu designer to create a standard Help menu that contains an **About** menu item, JADE displays a message box at run time if you do not set the **Application** class **aboutForm** property. This message box displays the following information.

```
----- Title:
"About Application:  application-name"

----- Contents:
Application:  application-name

Release: application-release-property

Jade Version: nn.nn.nn
```

This chapter covers the following topics.

- [Overview](#)
- [Structure of the Application Configuration File](#)
 - [Elements in the Application Configuration File](#)
- [Structure of the Web Services Consumer Configuration File](#)
 - [Elements in the Web Services Consumer Configuration File](#)
- [Creating and Maintaining the Configuration Files](#)
- [Using the Web Configuration Application](#)
 - [Web Configuration Application File Menu](#)
- [Web Application Configuration Examples](#)
 - [Configuration 1: Minimum](#)
 - [Configuration 2: Parallel Requests](#)
 - [Configuration 3: Multiple Connection Groups](#)
 - [Configuration 4: Multiple JADE Nodes](#)
 - [Configuration 5: Multiple Application Copies in Multiple Nodes](#)
 - [Configuration 6: Multiple JADE Servers](#)
 - [Configuration 7: Multiple Web Servers](#)

Overview

You can use an XML-based configuration file to define runtime configurations for Web-enabled applications, Web services consumers, and REST services applications. The advantages of using XML instead of standard initialization file syntax are as follows.

- Flexibility; for example, nesting is allowed
- Standards-based
- Development tools such as .NET primarily uses XML-based configuration files

If the [\[WebOptions\]](#) section in the JADE initialization file has a reference to the XML-based configuration file, the settings in the XML-based configuration file are used. If not, the settings in the [\[WebOptions\]](#) section are used.

Note There are additional runtime configuration parameters that you can specify. These can be defined only by using the XML format configuration file.

You can define two types of configuration file: one for Web-enabled applications (JADE forms, HTML documents, and Web service providers) and REST services applications, and one for Web service consumers.

- For Web-enabled applications and REST services applications, the JADE initialization file setting required to use the XML-based configuration file is as follows.

```
[WebOptions]
ApplicationConfigFile=file-name
```

- For Web service consumers, the JADE initialization file setting required to use the XML-based configuration file is as follows.

```
[WebOptions]
ConsumerConfigFile=file-name
```

For details about the [ApplicationConfigFile](#) and [ConsumerConfigFile](#) initialization file parameters, see the *JADE Initialization File Reference*.

Structure of the Application Configuration File

The following XML document represents a blank configuration file for a Web-enabled application.

```
<?xml version="1.0"?>
<jade_config>
  <application schema="" name="" id="">
    <web_config>
      <connection_name/>
      <protocol_family/>
      <application_copies/>
      <session_timeout/>
      <minimum_response_time/>
      <disable_messages/>
      <output_maximum_length/>
      <log_file_name/>
      <logmessagecontent/>
      <disable_logging/>
      <lock_retries/>
      <prompt_on_shutdown/>
      <firewall/>
      <monitor_font/>
      <base_uri>
        <protocol/>
        <machine_name/>
        <virtual_directory/>
      </base_uri>
      <support_library/>
      <jade_forms>
        <physical_directory/>
        <maximum_HTML_size/>
        <scrolling_text/>
        <show_modal/>
        <cross_browser/>
        <form_style/>
        <use_html4/>
        <web_events>
          <control_name/>
```

```
        </web_events>
        <image_type/>
        <page_sequencing/>
    </jade_forms>
    <html_documents>
        <home_page/>
        <html_page_sequencing/>
    </html_documents>
    <web_services_provider>
        <read_timeout/>
        <use_session_handling/>
    </web_services_provider>
</web_config>
</application>
</jade_config>
```

The empty XML elements represent parameters that you can configure; for example, you can configure the **<connection_name>** element with the value **WebApp**, by replacing the empty element with a non-empty element, as follows.

```
<connection_name>WebApp</connection_name>
```

In addition, the application XML element has three empty attributes that you can configure.

```
<application schema="BankingSchema" name="global" id="">
```

Elements in the Application Configuration File

The XML elements in the application configuration file are described in the following subsections.

jade_config element

The **jade_config** XML element is the root element for the document.

application element

The **application** XML element contains the following attributes.

- **schema** specifies the schema name
- **name** specifies the name of the Web-enabled application to which the settings apply
- **id** specifies an identifier string that enables copies of the same Web-enabled application to run with different configuration parameters

The application type can be a **JadeForms** application, an **HTMLDocuments** application, a **WebServices** provider application, or a **RestServices** application. For details, see ["Connecting to JADE Applications from Internet Information Server \(IIS\)"](#) or ["Connecting to JADE Applications from an Apache HTTP Server"](#), in Chapter 2 of the *JADE Installation and Configuration Guide*.

You can repeat the **application** element to specify configuration information for a number of applications. The special name **global** is used for configuration information that applies to all applications.

In the following example, there is configuration that applies globally to all applications, specific configuration information for **WebApp1**, and specific configuration information for **WebApp2**.

```
<?xml version="1.0"?>
<jade_config>
```

```
<application schema="WebAppSchema" name="global" id="">
    // configuration info applying to all applications
</application>
<application schema="WebAppSchema" name="WebApp1" id="">
    // configuration info specific to WebApp1
</application>
<application schema="WebAppSchema" name="WebApp2" id="">
    // configuration info specific to WebApp2
</application>
</jade_config>
```

The **id** attribute is used to enable the same application to be run with different configuration settings. In the following example, the **WebApp** application can be run with the **id="200"** configuration settings or the **id="300"** configuration settings.

```
<?xml version="1.0"?>
<jade_config>
    <application schema="WebAppSchema" name="WebApp" id="200">
        // configuration info specific to WebApp run with id "200"
    </application>
    <application schema="WebAppSchema" name="WebApp" id="300">
        // configuration info specific to WebApp run with id "200"
    </application>
</jade_config>
```

The following shortcut runs the **WebApp** application with the **id="300"** configuration settings.

```
jade.exe path=d:\webapp ini=c:\jade\system\jade.ini schema=WebAppSchema app=WebApp
startAppParameters id=300
```

web_config element

The **web_config** XML element is the root element for the general Web-enabled application settings.

connection_name element

The **connection_name** XML element specifies the name of a named pipe name or the TCP/IP address of the host system. The TCP/IP address is of the form *host:port number*, as shown in the following example.

```
<connection_name>168.212.226.204:50000</connection_name>
```

application_copies element

The **application_copies** XML element specifies the number of copies of the application to start up on initiation. The default value is **1**.

In the following example, five copies of the application would be started up. All copies would use the same IP port number, as specified in the **connection_name** element.

```
<application_copies>5</application_copies>
```

protocol_family

The **protocol_family** XML element specifies the TcpIp scheme that the Web server uses. The allowed values are **TcpIP**, **TcpIPv4**, **TcpIPv6**, or **TcpIPAny**. (**TcpIp** is a synonym for **TcpIpAny**.)

session_timeout element

The **session_timeout** XML element is the period in minutes after which a Web session is deleted if there has been no activity for that session. The default value of zero (0) indicates that the session does not time out. This setting is ignored if session handling is not turned on.

Web sessions are optional for Web services and do not apply to REST services.

minimum_response_time element

The **minimum_response_time** XML element specifies the maximum time in seconds before a response must be sent back to the requestor. The default of zero (0) indicates that there is no minimum response time.

disable_messages element

The **disable_messages** XML element specifies that messages will not be displayed in message boxes on the client workstation running the application. An example of a message box being displayed is a user, accessing a **JadeForms** application through a browser, making a repeat request. A message box warns that the repeated message will be ignored.

This element applies only to Web-enabled applications run as GUI applications. The default value for this element is **false**.

output_maximum_length element

The **output_maximum_length** XML element specifies the maximum number of characters to be displayed in the Web-enabled application monitor window. If you do not specify a maximum length for output or specify a value of zero (0), there is no restriction on the amount of text that is output.

log_file_name element

The **log_file_name** XML element specifies the file to which all logging that is displayed on the Web-enabled application monitor window is written. If the file name is invalid or the file could not be created, a file with the default name of **websessions.log** is used. If you do not specify a log file name, no logging to disk takes place.

logmessagecontent element

The **logmessagecontent** XML element specifies whether Web message content is logged in the JADE Web Application Monitor window. By default, any Web logging includes the **Query String = content** and **Http String = content** output. The default value for this element is **true**.

Set this element to **false** if you do not want to include the **Query String = content** and **Http String = content** message content.

This element applies only if the **disable_logging** element in the Web application configuration file or the **DisableLogging** parameter in the [WebOptions] section of the JADE initialization file is set to **true**, or tracing is turned on when the **Enable Logging** command is displayed in the View menu of the JADE Web Application Monitor window. (The value of this command toggles between **Enable Logging** and **Disable Logging**.)

When you initiate the Web Application monitor and tracing is on, a message is displayed indicating the status of the **LogMessageContent** element, as follows.

```
Message content logging is enabled|disabled
```

disable_logging element

The **disable_logging** XML element specifies whether the JADE Monitor application logs and displays information relating to requests and whether responses for Web-enabled applications run as GUI applications. Setting a value for this element has no impact on Web-enabled non-GUI applications. Disabling logging has a significant positive impact on throughput.

Tip Use Web-enabled non-GUI applications for production systems and Web-enabled GUI applications for debugging purposes.

The default value for this element is **false**.

lock_retries element

The **lock_retries** XML element specifies the number of times to retry a lock exception before aborting a request. The default value for this element is **20**.

prompt_on_shutdown element

The **prompt_on_shutdown** XML element specifies whether a message is displayed when the application is shut down and there are active Web sessions. Setting a value for this element has no impact on Web-enabled non-GUI applications.

If the value of the **prompt_on_shutdown** element is **true** (which is the default value), a message is displayed and the application does not shutdown until a response is made to the message.

firewall element

The **firewall** XML element specifies whether the **jadehttp** support library is outside a firewall.

If the value of the **firewall** element is **true**, any images that are generated must be transferred to **jadehttp** so that it can create the files in a directory outside the firewall (as the Web-enabled application is inside the firewall, it cannot see the directory).

The default value for this element is **false**.

monitor_font element

The **monitor_font** XML element specifies the font used in the Web Application Monitor window. The font is specified as a comma-delimited string specifying the font name, font size, font bold, and font color respectively.

The following example displays monitor output in red using the 15 point bold Tahoma font.

```
<monitor_font>Tahoma,15,true,255</monitor_font>
```

base_uri element

The **base_uri** XML element is the root element for specifying the various parts of the Uniform Resource Identifier (URI).

protocol element

The **protocol** XML element specifies the default protocol to be used. The value of this element can be **http** (which is the default value) or **https**.

The values of the **protocol**, **machine_name**, and **virtual_directory** elements are used for generating hyperlinks and the form action value.

machine_name element

The **machine_name** XML element specifies the machine name or TCP/IP address of the target host. The default value for this element is null.

The values of the **protocol**, **machine_name**, and **virtual_directory** XML elements are used for generating hyperlinks and the form action value.

virtual_directory element

The **virtual_directory** XML element is the directory as specified to the Web server and it is the directory where the **jadehttp** module is located. The default value for this element is null.

The values of the **protocol**, **machine_name**, and **virtual_directory** XML elements are used for generating hyperlinks and the form action value.

support_library element

The **support_library** XML element is the library or module used for communicating between the Web server and the JADE Web-enabled application.

jade_forms element

The **jade_forms** XML element is the root element for specifying configuration parameters that apply only to **JadeForms** applications.

physical_directory element

The **physical_directory** XML element specifies the physical directory that corresponds to the virtual directory specified by the value of the **virtual_directory** element. Image and script files that are generated by the framework are placed in this directory. The default value for this element is null.

maximum_HTML_size element

The **maximum_HTML_size** XML element specifies the maximum length of the HTML string that is to be generated. The default value for this element is zero (**0**), which means there is no limit.

scrolling_text element

The **scrolling_text** XML element specifies the scrolling text that is to appear in the status window of the browser. The default value for this element is null.

show_modal element

The **show_modal** XML element specifies whether an instruction in the code to display a form by using the **showModal** method actually displays the form by using the **show** method, or whether an error is generated.

The default value for this element is **false**, which results in an error being generated.

cross_browser element

The **cross_browser** XML element specifies whether cross-browser compatibility is required. If you set the value of this element to **true**, the generated code for all browsers is HTML version 3.2. The default value for this element is **false**.

form_style element

The **form_style** XML element specifies whether Web pages are displayed as windows (that is, with captions and borders) when using Internet Explorer browsers or simply as Web browser pages.

The default value for this element is **false**.

use_html4 element

The **use_html4** XML element specifies whether HTML 4.0 generation is required for all browsers. The default value for this element is **false**.

web_events element

The **web_events** XML element is the root element that enables you to specify (for Internet Explorer 4.0 and above only) whether events other than clicking a submit button or a hyperlink can be captured.

control_name element

The **control_name** XML element specifies controls that can participate in a **click** event other than the standard default values. Separate control names with commas.

image_type element

The **image_type** XML element is the format of the image files that are displayed on the generated Web pages. The default generated image type is **jpg** (Joint Photographic Experts Group). You can also specify a **png** (Portable Network Graphics) or **gif** (Graphics Interchange Format) file type.

page_sequencing element

The **page_sequencing** XML element specifies that forms that are generated have a hidden field with a sequence number, which is incremented with each request. If the value of this element is set to **true**, an exception is raised if requests arrive in an incorrect sequence. The exception must be handled by user code. The default value for this element is **false**.

html_documents element

The **html_documents** XML element is the root element that enables you to specify configuration parameters that apply only applications of **HTMLDocuments** type.

home_page element

The **home_page** XML element specifies the first page to be displayed when a request is made to the application. The value of this element overrides the value set for the application. The default value for this element is null.

html_page_sequencing element

The **html_page_sequencing** XML element specifies that generated forms have a hidden field with a sequence number, which is incremented with each request. If the value of this element is set to **true**, an exception is raised if requests arrive in an incorrect sequence. The exception must be handled by user code. The default value for this element is **false**.

web_services_provider element

The **web_services_provider** XML element is the root element for specifying configuration parameters that apply only to Web services provider applications.

read_timeout element

The **read_timeout** XML element is the length of time in seconds to wait before terminating a read request. The default value for this element is **120**.

use_session_handling element

The **use_session_handling** XML element to specify that a SOAP header is to be generated automatically with a session id for every request in a Web services application.

Structure of the Web Services Consumer Configuration File

The following XML document represents a blank configuration file for a Web-enabled application.

```
<?xml version="1.0"?>
<jade_config>
  <web_services_consumer schema="" name="" id="">
    <web_config>
      <consumer>
        <endpoint></endpoint>
        <maximum_connections></maximum_connections>
        <connection_timeout></connection_timeout>
        <send_timeout></send_timeout>
        <receive_timeout></receive_timeout>
      </consumer>
    </web_config>
  </web_services_consumer>
</jade_config>
```

The empty XML elements represent parameters that can be configured; for example, you can configure the **<connection_name>** element with the value **WebApp**, by replacing the empty element with a non-empty element, as follows.

```
<connection_name>WebApp</connection_name>
```

In addition, the application XML element has three empty attributes that you can configure, as follows.

```
<application schema="WebConsumer" name="global" id="200">
```

Elements in the Web Services Consumer Configuration File

The XML elements in the Web services consumer configuration file are described in the following sections.

jade_config element

The **jade_config** XML element is the root element for the document.

web_services_consumer element

The **web_services_consumer** XML element contains the following attributes.

- **schema** specifies the schema name
- **name** specifies the name of the Web services consumer to which the settings apply
- **id** specifies an identifier string that enables copies of the same Web services consumer application to run with different configuration parameters

You can repeat the **web_services_consumer** element to specify configuration information for a number of Web services consumers. The special name **global** is used for configuration information that applies to all Web services consumers. Only settings that need to be changed are defined in a specific consumer section.

In the following example, there is configuration information that applies globally to all Web services consumers, specific configuration information for **WebConsumer1**, and specific configuration information for **WebConsumer2**.

```
<?xml version="1.0"?>
<jade_config>
  <web_services_consumer schema="WebSchema" name="global" id="">
    // configuration info applying to all consumer applications
  </web_services_consumer>
  <web_services_consumer schema="WebSchema" name="WebConsumer1" id="">
    // configuration info specific to WebConsumer1
  </web_services_consumer>
  <web_services_consumer schema="WebSchema" name="WebConsumer2" id="">
    // configuration info specific to WebConsumer2
  </web_services_consumer>
</jade_config>
```

The **id** attribute is used to enable the same Web services consumer to be run with different configuration settings. In the following example, the application **WebConsumer** can be run with the **id="200"** configuration settings or the **id="300"** configuration settings.

```
<?xml version="1.0"?>
<jade_config>
  <web_services_consumer schema="WebSchema" name="WebConsumer" id="200">
    // configuration info specific to WebConsumer run with id "200"
  </web_services_consumer>
  <web_services_consumer schema="WebSchema" name="WebConsumer" id="300">
    // configuration info specific to WebConsumer run with id "300"
  </web_services_consumer>
</jade_config>
```

The following shortcut runs the **WebConsumer** application with the **id="300"** configuration settings.

```
jade.exe path=d:\webapp ini=c:\jade\system\jade.ini schema=WebSchema
app=WebConsumer startAppParameters id=300
```

web_config element

The **web_config** XML element is the root element for the general Web-enabled application settings.

consumer element

The **consumer** XML element is the root element for the general Web consumer application settings.

endpoint element

The **endpoint** XML element specifies the endpoint Uniform Resource Locator (URL) to be used when calling this consumer Web service.

maximum_connections element

The **maximum_connections** XML element specifies the maximum number of allowed simultaneous connections. The default value for this element is two (2), which comes from the HTTP 1.1 standard.

For example, setting the value of the **maximum_connections** XML element to eight (8) enables a maximum of eight (8) simultaneous connections on the Windows platform.

connection_timeout element

The **connection_timeout** XML element specifies the length of time in seconds to wait before terminating a connection attempt. The default value for this element is **120**.

send_timeout element

The **send_timeout** XML element specifies the length of time in seconds to wait before terminating a send attempt. The default value for this element is **120**.

receive_timeout element

The **receive_timeout** XML element specifies the length of time in seconds to wait before terminating a receive attempt. The default value for this element is **120**.

Creating and Maintaining the Configuration Files

The Web Configuration application enables you to create and maintain the XML configuration files for Web-enabled applications, REST services applications, and Web service consumers, by entering values for the elements on a form.

Using the Web Configuration Application

You can create and maintain configuration files using any text editor (for example, **Notepad**).

The following XML document represents a blank configuration file for a Web-enabled application.

```
<?xml version="1.0"?>
<jade_config>
  <application schema="" name="" id="">
    <web_config>
      <connection_name/>
```

```

    <application_copies/>
    <session_timeout/>
    <minimum_response_time/>
    <disable_messages/>
    <log_file_name/>
    <disable_logging/>
    <lock_retries/>
    <prompt_on_shutdown/>
    <firewall/>
    <base_uri>
        <protocol/>
        <machine_name/>
        <virtual_directory/>
    </base_uri>
    <support_library/>
    <jade_forms>
        <physical_directory/>
        <maximum_HTML_size/>
        <scrolling_text/>
        <show_modal/>
        <cross_browser/>
        <form_style/>
        <use_html4/>
        <web_events>
            <control_name/>
        </web_events>
        <image_type/>
        <page_sequencing/>
    </jade_forms>
    <html_documents>
        <home_page/>
        <html_page_sequencing/>
    </html_documents>
    <web_services_provider>
        <read_timeout/>
    </web_services_provider>
</web_config>
</application>
</jade_config>

```

The empty elements represent parameters that you can configure; for example, you can configure the **<connection_name/>** element with the value **WebApp**, by replacing the empty element with a non-empty element, as follows.

```
<connection_name>WebApp</connection_name>
```

In addition, the application element has three empty attributes that you can configure, as follows.

```
<application schema="BankingSchema" name="global" id="">
```

Tip You do not have to edit the XML directly, as JADE provides the Web Configuration application that enables you to create and maintain the configuration files.

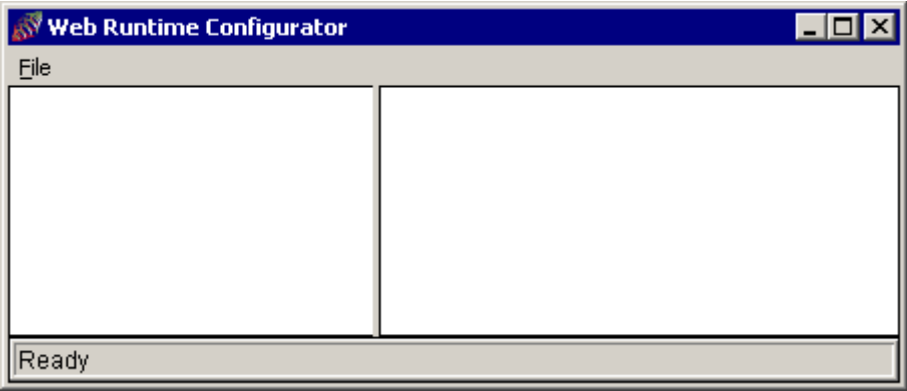
The shortcut to run the Web Configuration application has the schema parameter set to **JadeMonitorSchema** and the application parameter set to **JadeWebConfigurator**.

You can pass an optional **filename** command line parameter to specify the name of the file to edit.

The **filename** parameter must be separated from the standard parameters by the **startAppParameters** parameter, as follows.

```
c:\jade\bin\jade.exe schema=JadeMonitorSchema
                        app=JadeWebConfigurator
                        path=c:\jade\system
                        ini=c:\jade\system\jade.ini
                        startAppParameters
                        filename=c:\temp\webservice.xml
```

Running the application (without **startAppParameters** and **filename** parameters) displays the following window.



Web Configuration Application File Menu

The File menu in the Web Runtime Configurator window provides the following commands.

Command	Description
New	Provides a submenu containing the Application and Consumer commands
Application	Enables you to configure a Web-enabled or REST services application
Consumer	Enables you to configure a Web services consumer
Open	Displays the common File Open dialog
Append New	Defines multiple Web applications or consumers in the one configuration file
Save	Displays the common File Save dialog
Save As	Displays the common Save As dialog
Exit	Exits from the Web Configuration application

For details, see the following subsections.

New Command

- » **To configure a new Web application or Web service consumer**
 - Select the **New** command from the Web Runtime Configurator File menu.

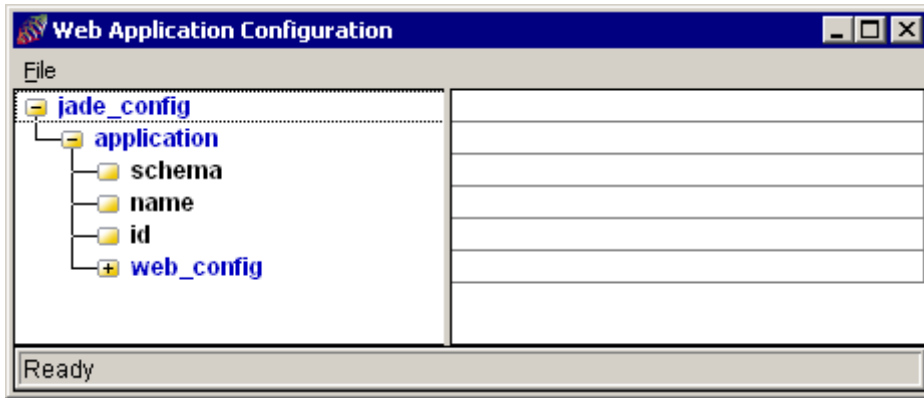
A submenu containing the **Application** and **Consumer** commands is then displayed. For details, see the following subsections.

Application Command

» To configure a new Web application

1. Select the **Application** command from the **New** command submenu.

The list in the pane at the left of the Web Runtime Configurator window is then populated with items in blue and items in black, as shown in the following image.



The black items (for example, **schema**, **name**, and **id**) represent configuration parameters. You can enter a value for a configuration parameter in the corresponding cell in the pane at the right of the window.

The blue items (for example, **jade_config**, **application**, and **web_config**) represent groupings of configuration parameters. When you click the plus sign (+) to the left of a blue item, the display expands to show the configuration parameters of the item and possibly other grouping items.

For details about the elements in a Web application configuration file that you can configure, see "[Elements in the Application Configuration File](#)", earlier in this chapter.

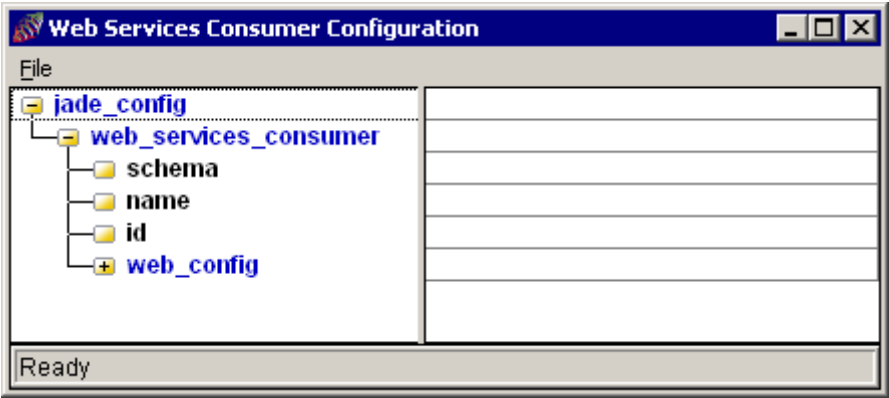
2. Expand the node in the pane at the left of each configuration file entity whose values you want to display in the pane at the right of the window.
3. To edit the configuration file, click in the appropriate row in the pane at the right of the window and maintain the value in that row to meet your requirements.
4. Repeat step 3 of this instruction for each value that you want to maintain.
5. Save the file.

Consumer Command

» To configure a new Web service consumer

- 1. Select the **Consumer** command from the **New** command submenu.

The list in the pane at the left of the Web Runtime Configurator window is then populated with items in blue and items in black, as shown in the following image.



The black items (for example, **schema**, **name**, and **id**) represent configuration parameters. You can enter a value for a configuration parameter in the corresponding cell in the pane at the right of the window.

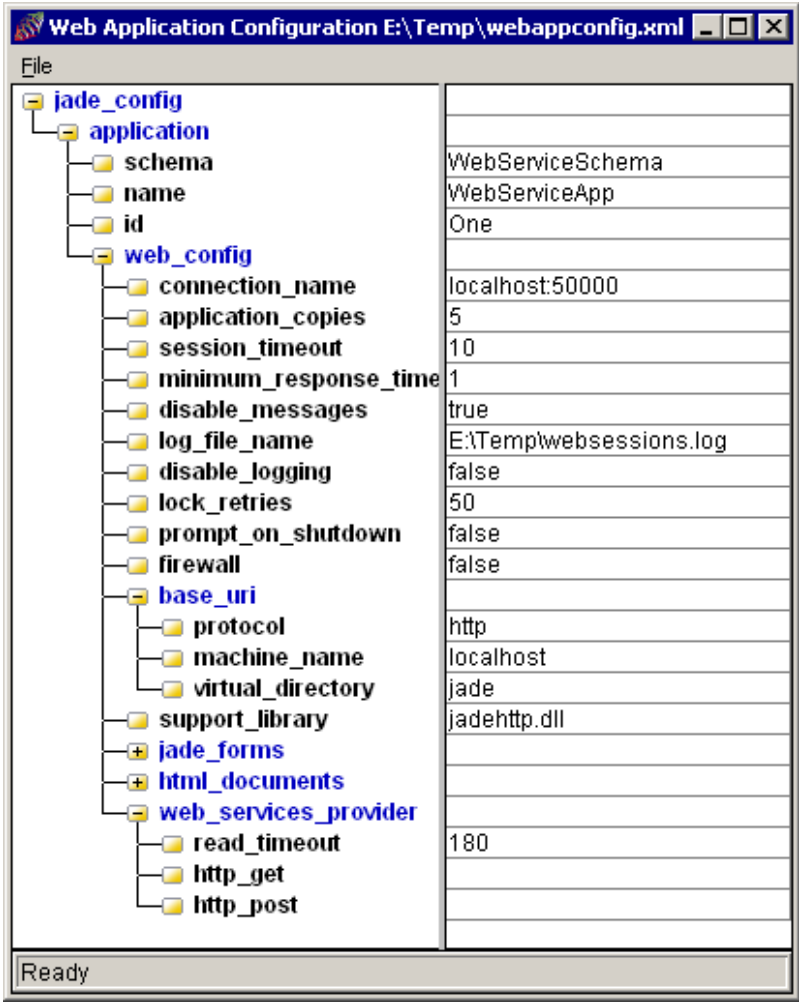
The blue items (for example, **jade_config**, **application**, and **web_config**) represent groupings of configuration parameters. When you click the plus sign (+) to the left of a blue item, the display expands to show the configuration parameters of the item and possibly other grouping items.

For details about the elements in a Web services consumer configuration file that you can configure, see "[Elements in the Web Services Consumer Configuration File](#)", earlier in this chapter.

Open Command

» **To edit an existing Web application configuration file or Web services consumer configuration file**

1. Select the **Open** command from the Web Runtime Configurator File menu.
2. In the common File Open dialog that is then displayed, select the configuration file that you want to edit. The pane at the right of the Web Runtime Configurator window is then populated with the appropriate information, as shown in the example in the following image in which a file called **e:\temp\webappconfig.xml** was selected.

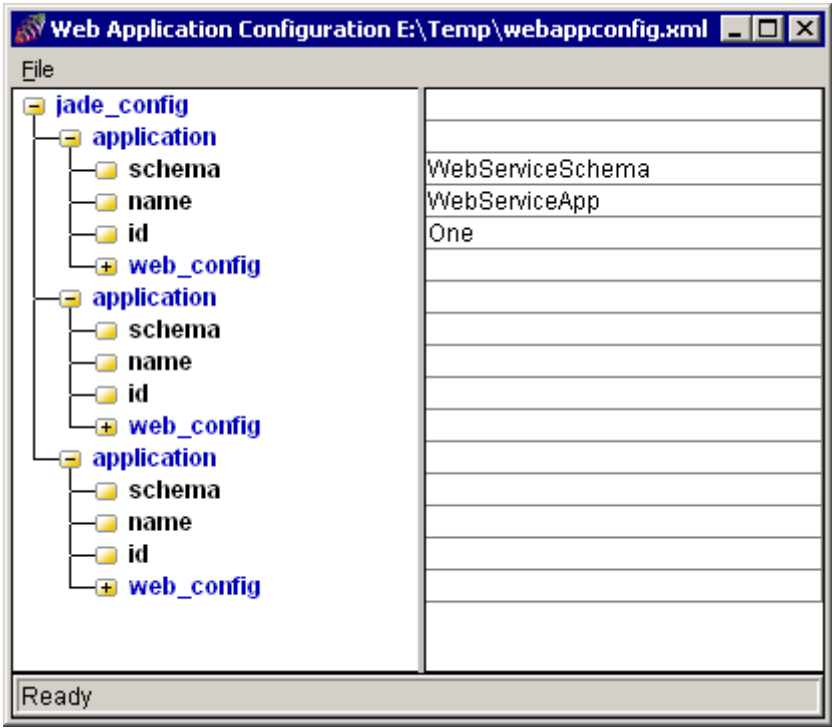


3. Expand the node in the pane at the left of each configuration file entity whose values you want to display in the pane at the right of the window.
4. To edit the configuration file, click in the appropriate row in the pane at the right of the window and maintain the value in that row to meet your requirements.
5. Repeat step 4 of this instruction for each value that you want to maintain.
6. Save the file.

Append New Command

» To define multiple Web applications or consumers in one configuration file

1. Select the **Append New** command from the Web Runtime Configurator File menu. For example, to add two more applications to a file called `e:\temp\webappconfig.xml`, select the **Append New** command twice, in which case the example shown in the following image is then displayed.



2. Expand the node in the pane at the left of each Web application configuration file entity (or Web services consumer configuration file) whose values you want to display in the pane at the right of the window.
3. To edit the configuration files, click in the appropriate row in the pane at the right of the window and maintain the value in that row to meet your requirements.
4. Repeat step 3 of this instruction for each value that you want to maintain.
5. Use the File menu **Save As** command to save the file to an `.xml` file name and location of your choice.

Save Command

» To save your Web application or Web service consumer configuration file

- Select the **Save** command from the File menu.

The common File Save dialog is then displayed, enabling you to save your `.xml` file.

Save As Command

» To save your configuration file to a specified name and location

- Select the **Save As** command from the File menu.

The common Save As dialog is then displayed, enabling you to save your **.xml** file to a name and location of your choice.

If you are editing an existing file, the file name defaults to the name of this file. If it is not an existing file, specify the file name that you require.

Exit Command

Select the **Exit** command from the Web Runtime Configurator File menu to shut down the Web Configuration application and close the Web Runtime Configurator window.

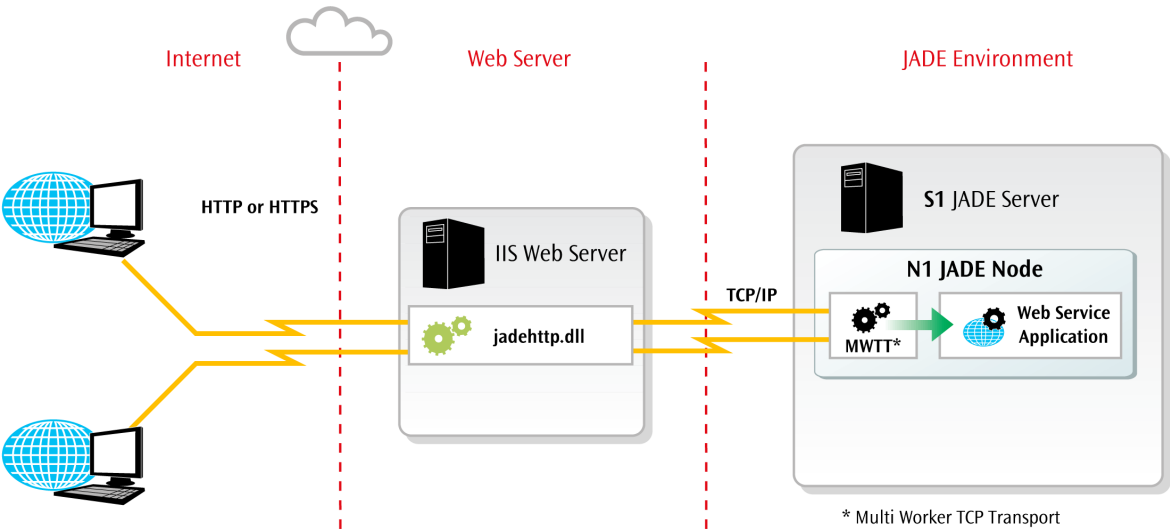
Web Application Configuration Examples

The following discussion of Web application configurations assumes that a JADE Web service application named **MyWebService** is defined in the **MySchema** schema.

The sections of the **jadehttp.ini** file and the Web application configuration file included with each configuration are incomplete. They include only the elements required to illustrate each configuration; you may require additional elements to build a running environment.

For examples, see the following subsections.

Configuration 1: Minimum



This image shows the simplest configuration. The Web service is offered to consumers at a single URL (<http://www.Company.com/Jade/jadehttp.dll?MyWebService...>) supplied by **IISWebServer-1**. A single copy of the Web application running in **JadeNode-N1** on **JadeServer-S1** handles all requests, one at a time.

The TCP port number used by **jadehttp.dll** to connect to the Multi Worker TCP Transport (MWTT) group of the application is 21001 (**TcpConnection** and **TcpPort** parameter values). A maximum of 10 outstanding requests is permitted (**MaxInUse** parameter value).

The **jadehttp.dll** queues a request to the Web application by opening a new connection or by reusing an idle connection. When the **jadehttp.dll** has received the reply from a connection, it adds the connection to the idle list. In this case, up to ten connections can be open between the **jadehttp.dll** and the MWTT for the Web application. When ten requests are outstanding (and therefore ten connections are in use), additional requests are rejected with an *application is busy* error.

If you specify a host name for the **TcplpConnection** parameter, all DNS-provided address will be attempted. Both the **TcpIPv6** and **TcpIPv4** protocols will be attempted on the provided IP addresses. Each connection failure will be logged, and the next available combination tried.

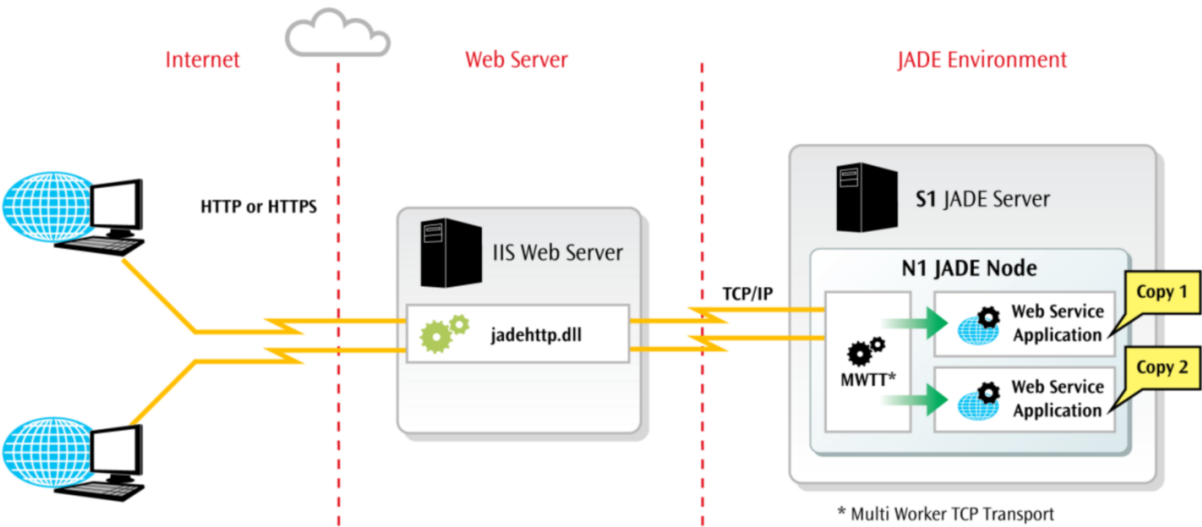
Configuration 1 - JadeHttp.ini File

```
[MyWebService]
ApplicationType = WebServices
TcpConnection = JadeServer-S1
TcpPort = 21001
MaxInUse = 10
```

Configuration 1 - Web Application Configuration File

```
<jade_config>
  <application schema="MySchema" name="MyWebService" id="Group-M1">
    <web_config>
      <connection_name>JadeServer-S1:21001</connection_name>
      <application_copies>1</application_copies>
      <base_uri>
        <protocol>http</protocol>
        <machine_name>www.Company.com</machine_name>
        <virtual_directory>Jade</virtual_directory>
      </base_uri>
      <web_services_provider/>
    </web_config>
  </application>
</jade_config>
```

Configuration 2: Parallel Requests



This image shows a configuration that splits the request processing workload between two copies of the Web application running in a single JADE node. A maximum of 20 outstanding requests is permitted (**MaxInUse**). When an application copy completes a request on a connection, it selects the next queued connection from the MWT connection queue and reads and processes the request. Each application copy is a JADE process and requires a process license.

You can run more application copies in parallel, which will reduce the average overall elapsed time for each request but increase contention for resources.

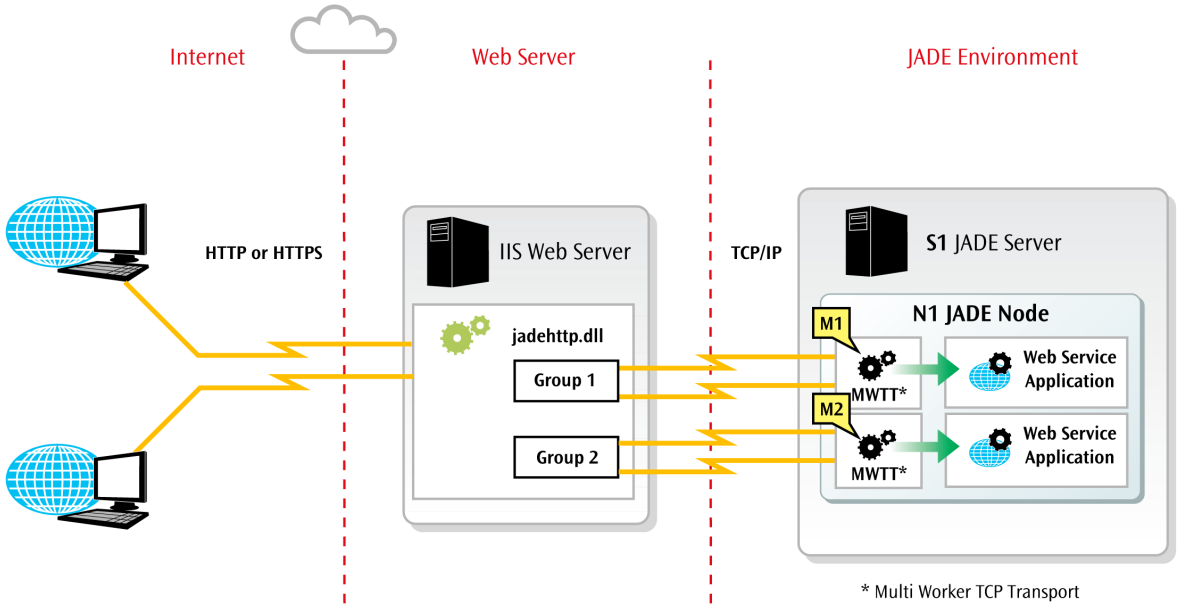
Configuration 2 - JadeHttp.ini File

```
[MyWebService]
ApplicationType = WebServices
TcpConnection = JadeServer-S1
TcpPort = 21001
MaxInUse = 20
```

Configuration 2 - Web Application Configuration File

```
<jade_config>
  <application schema="MySchema" name="MyWebService" id="Group-M1">
    <web_config>
      <connection_name>JadeServer-S1:21001</connection_name>
      <application_copies>2</application_copies>
      <base_uri>
        <protocol>http</protocol>
        <machine_name>www.Company.com</machine_name>
        <virtual_directory>Jade</virtual_directory>
      </base_uri>
      <web_services_provider/>
    </web_config>
  </application>
</jade_config>
```

Configuration 3: Multiple Connection Groups



This image shows a configuration that splits the request processing workload between two independent copies of a Web application. Each copy has a separate independent connection queue. The **jadehttp.dll** distributes requests by selecting a new or idle connection from the next group that has not yet reached **MaxInUse** connections (round-robin selection).

Two application sections (each with a unique id) are defined, because the MWTT associated with each Web application listens on a different TCP address and port.

It is unusual to use this configuration to expose a single Web application as it is possible that the request processing load is not equally shared between the Web application copies. Configuration 2 in the previous section is much more effective at sharing the processing load between multiple copies of a Web application.

The configuration in this section is more likely when several unrelated Web applications are exposed from the same database environment. In that case, the **jadehttp.ini** file will contain a separate section for each of the Web applications rather than the dual group section shown in the example in the following subsection.

Configuration 3 - JadeHttp.ini File

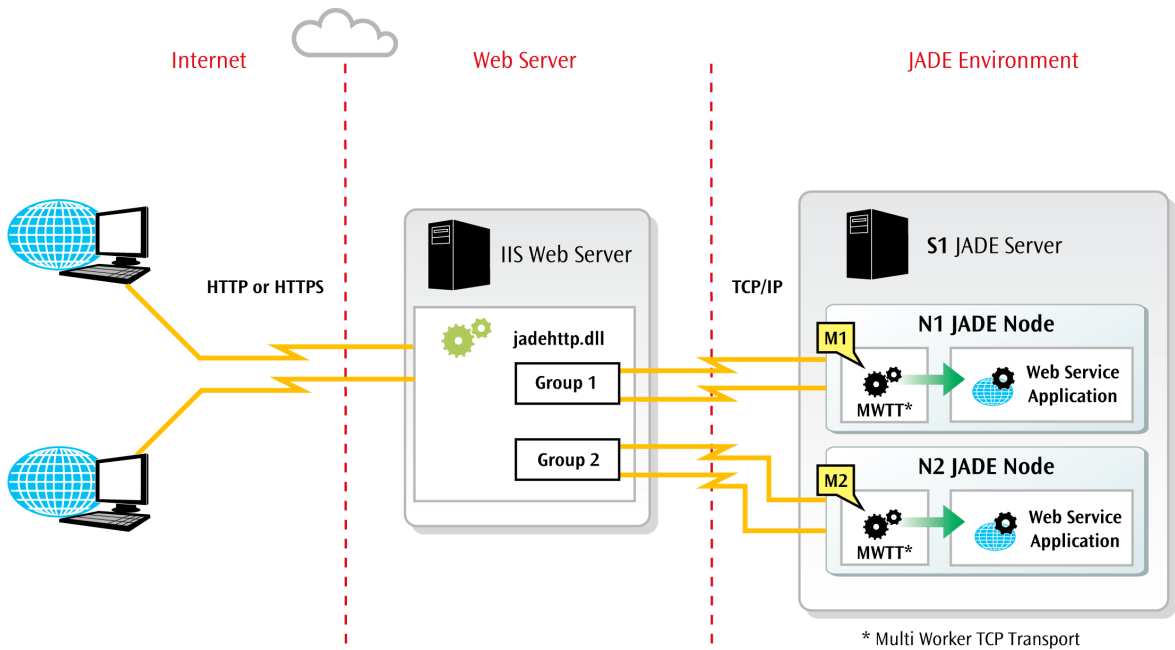
```
[MyWebService]
ApplicationType = WebServices
TcpConnection1= JadeServer-S1
TcpPort1 = 21001
MaxInUse1 = 20
TcpConnection2 = JadeServer-S1
TcpPort2 = 21002
MaxInUse2 = 20
```

Configuration 3 - Web Application Configuration File

```
<jade_config>
  <application schema="MySchema" name="MyWebService" id="Group-M1">
    <web_config>
```

```
<connection_name>JadeServer-S1:21001</connection_name>
<application_copies>1</application_copies>
<base_uri>
  <protocol>http</protocol>
  <machine_name>www.Company.com</machine_name>
  <virtual_directory>Jade</virtual_directory>
</base_uri>
<web_services_provider/>
</web_config>
</application>
<application schema="MySchema" name="MyWebService" id="Group-M2">
  <web_config>
    <connection_name>JadeServer-S1:21002</connection_name>
    <application_copies>1</application_copies>
    <base_uri>
      <protocol>http</protocol>
      <machine_name>www.Company.com</machine_name>
      <virtual_directory>Jade</virtual_directory>
    </base_uri>
    <web_services_provider/>
  </web_config>
</application>
</jade_config>
```

Configuration 4: Multiple JADE Nodes



This image shows a configuration that splits the request processing workload between two independent copies of a Web application, each running in separate JADE node within the same machine. This configuration avoids the contention issues that arise when multiple JADE processes execute within a node but it requires more resource such as memory.

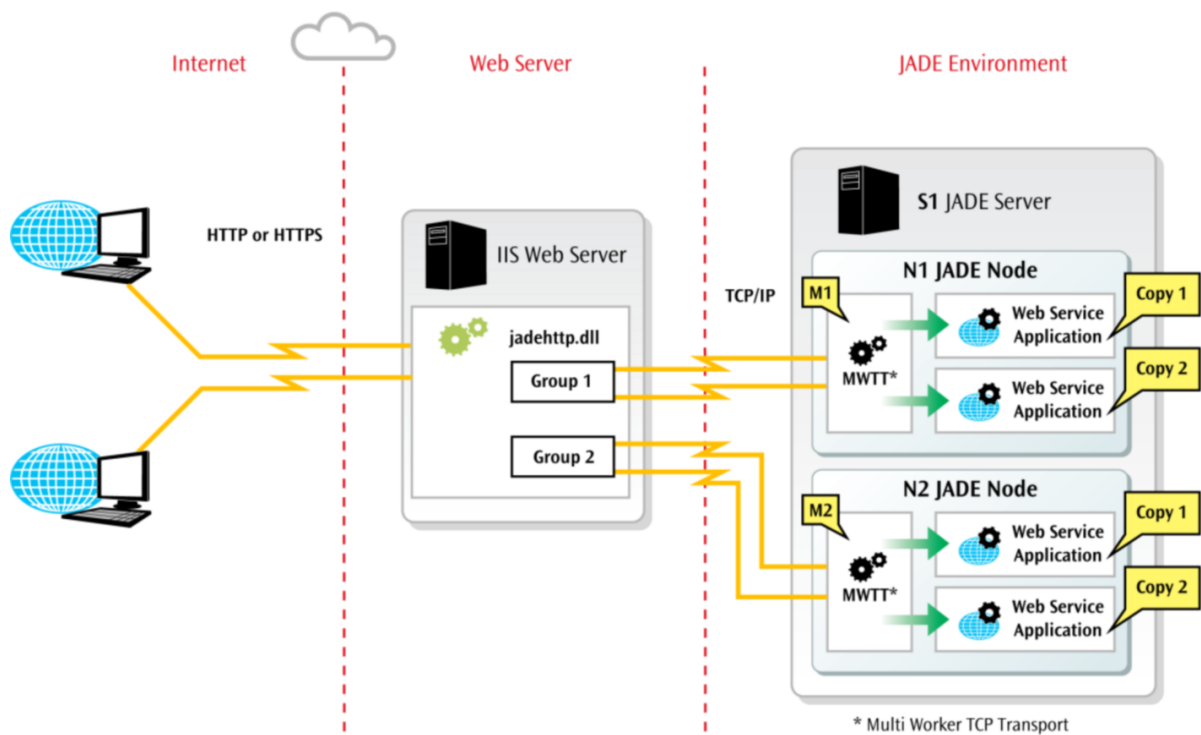
Configuration 4 - JadeHttp.ini File

```
[MyWebService]
ApplicationType = WebServices
TcpConnection1= JadeServer-S1
TcpPort1 = 21001
MaxInUse1 = 20
TcpConnection2 = JadeServer-S1
TcpPort2 = 21002
MaxInUse2 = 20
```

Configuration 4 - Web Application Configuration File

```
<jade_config>
  <application schema="MySchema" name="MyWebService" id="Group-M1">
    <web_config>
      <connection_name>JadeServer-S1:21001</connection_name>
      <application_copies>1</application_copies>
      <base_uri>
        <protocol>http</protocol>
        <machine_name>www.Company.com</machine_name>
        <virtual_directory>Jade</virtual_directory>
      </base_uri>
      <web_services_provider/>
    </web_config>
  </application>
  <application schema="MySchema" name="MyWebService" id="Group-M2">
    <web_config>
      <connection_name>JadeServer-S1:21002</connection_name>
      <application_copies>1</application_copies>
      <base_uri>
        <protocol>http</protocol>
        <machine_name>www.Company.com</machine_name>
        <virtual_directory>Jade</virtual_directory>
      </base_uri>
      <web_services_provider/>
    </web_config>
  </application>
</jade_config>
```

Configuration 5: Multiple Application Copies in Multiple Nodes



This image shows a configuration that splits the request processing workload between multiple copies of a Web application running in separate JADE nodes within the same machine. This configuration has less resource contention than a configuration that has all Web application copies running in the same node.

You can use this configuration when a large number of requests must be processed in parallel but the local resource usage (for example, CPU) per request is low, in which case the number of copies for each node will be much higher (for example, 30 to 40). In addition, the outstanding request limit will be high (in this example, 500 for each node).

Configuration 5 - JadeHttp.ini File

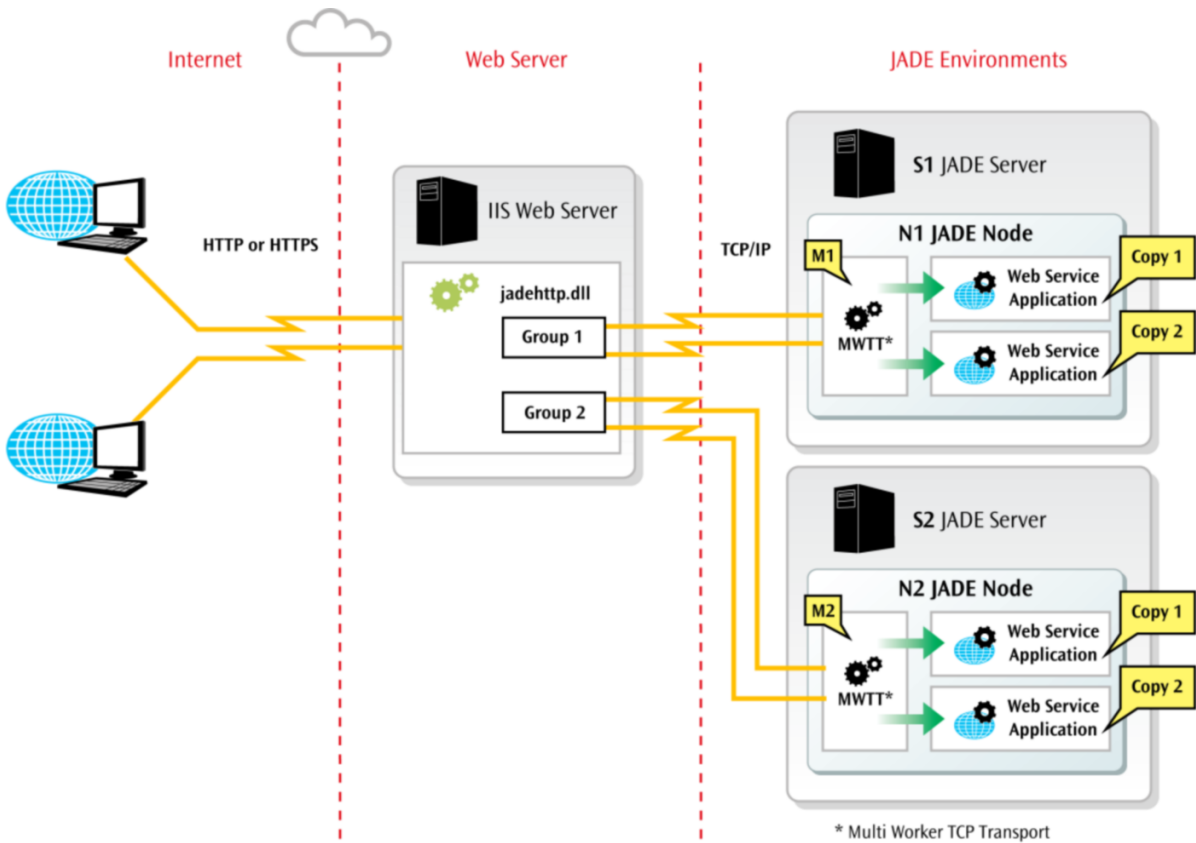
```
[MyWebService]
ApplicationType = WebServices
TcpConnection1= JadeServer-S1
TcpPort1 = 21001
MaxInUse1 = 500
TcpConnection2 = JadeServer-S1
TcpPort2 = 21002
MaxInUse2 = 500
```

Configuration 5 - Web Application Configuration File

```
<jade_config>
  <application schema="MySchema" name="MyWebService" id="Group-M1">
    <web_config>
      <connection_name>JadeServer-S1:21001</connection_name>
      <application_copies>2</application_copies>
```

```
<base_uri>
  <protocol>http</protocol>
  <machine_name>www.Company.com</machine_name>
  <virtual_directory>Jade</virtual_directory>
</base_uri>
<web_services_provider/>
</web_config>
</application>
<application schema="MySchema" name="MyWebService" id="Group-M2">
  <web_config>
    <connection_name>JadeServer-S1:21002</connection_name>
    <application_copies>2</application_copies>
    <base_uri>
      <protocol>http</protocol>
      <machine_name>www.Company.com</machine_name>
      <virtual_directory>Jade</virtual_directory>
    </base_uri>
    <web_services_provider/>"
  </web_config>
</application>
</jade_config>
```

Configuration 6: Multiple JADE Servers



This image shows a configuration that splits the request processing workload between two servers (that is, **S1** and **S2**). A single JADE node is running in each server. Although only two copies of the Web application are shown for each node, it is more usual to run two or three copies for each logical CPU, depending on the resource usage for each request. Each node (and therefore each server) can have up to 200 outstanding requests.

You could extend this configuration by running multiple nodes in one or more of the servers.

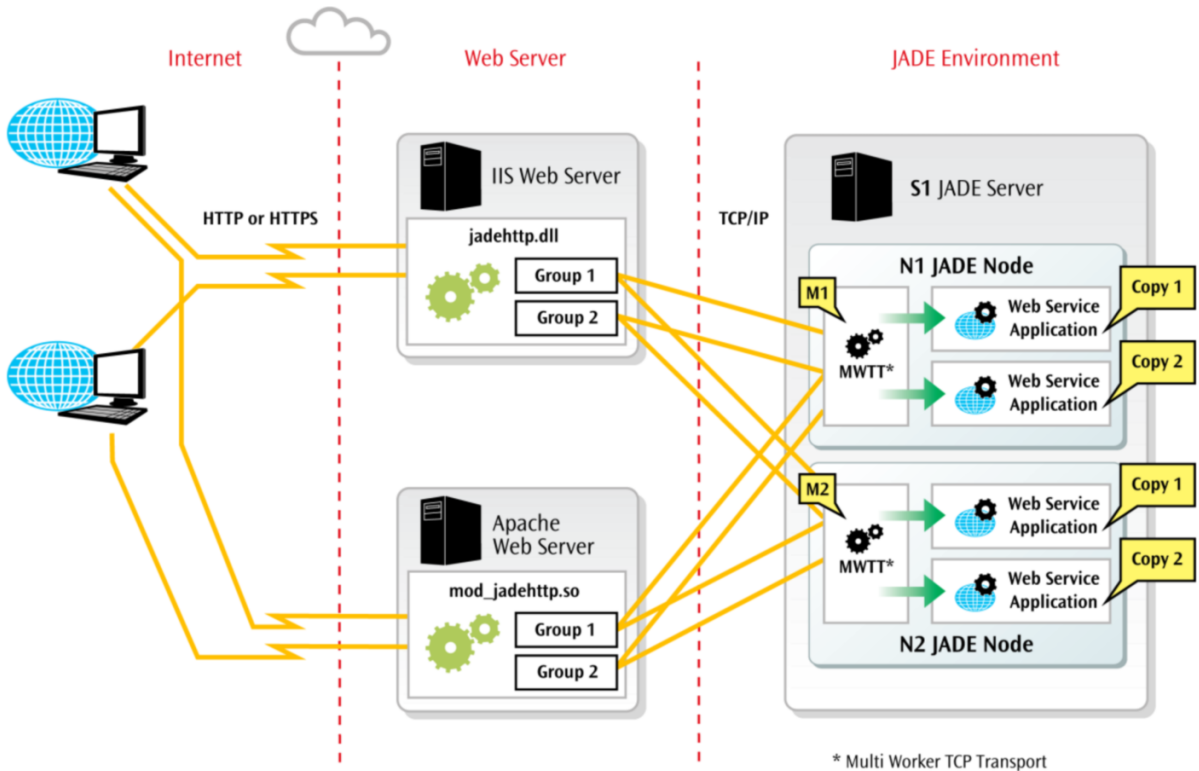
Configuration 6 - JadeHttp.ini File

```
[MyWebService]
ApplicationType = WebServices
TcpConnection1= JadeServer-S1
TcpPort1 = 21001
MaxInUse1 = 200
TcpConnection2 = JadeServer-S2
TcpPort2 = 21002
MaxInUse2 = 200
```

Configuration 6 - Web Application Configuration File

```
<jade_config>
  <application schema="MySchema" name="MyWebService" id="Group-M1">
    <web_config>
      <connection_name>JadeServer-S1:21001</connection_name>
      <application_copies>2</application_copies>
      <base_uri>
        <protocol>http</protocol>
        <machine_name>www.Company.com</machine_name>
        <virtual_directory>Jade</virtual_directory>
      </base_uri>
      <web_services_provider/>
    </web_config>
  </application>
  <application schema="MySchema" name="MyWebService" id="Group-M2">
    <web_config>
      <connection_name>JadeServer-S2:21002</connection_name>
      <application_copies>2</application_copies>
      <base_uri>
        <protocol>http</protocol>
        <machine_name>www.Company.com</machine_name>
        <virtual_directory>Jade</virtual_directory>
      </base_uri>
      <web_services_provider/>
    </web_config>
  </application>
</jade_config>
```

Configuration 7: Multiple Web Servers



This image shows a configuration that exposes the Web service via two Web servers. In this example, one is IIS and the other is Apache. A reason for using this configuration is to allow one Web server to expose the service within your company and the other to expose the service to the Internet.

Configuration 7 - JadeHttp.ini File

```
[MyWebService]
ApplicationType = WebServices
TcpConnection1= JadeServer-S1
TcpPort1 = 21001
MaxInUse1 = 200
TcpConnection2 = JadeServer-S1
TcpPort2 = 21002
MaxInUse2 = 200
```

Configuration 7 - Apache Configuration File

```
<Location /MyWebService>
# Apache directives
SetHandler jadehttp-handler
Order allow,deny
Allow from all
# Jade directives
Application MyWebService
ApplicationType WebServices
TcpConnection1 JadeServer-S1 21001 1 200
```

```
TcpConnection2 JadeServer-S1 21002 1 200
</Location>
```

Configuration 7 - Web Application Configuration File

```
<jade_config>
  <application schema="MySchema" name="MyWebService" id="Group-M1">
    <web_config>
      <connection_name>JadeServer-S1:21001</connection_name>
      <application_copies>2</application_copies>
      <base_uri>
        <protocol>http</protocol>
        <machine_name>www.Company.com</machine_name>
        <virtual_directory>MyWebService</virtual_directory>
      </base_uri>
      <web_services_provider/>
    </web_config>
  </application>
  <application schema="MySchema" name="MyWebService" id="Group-M2">
    <web_config>
      <connection_name>JadeServer-S1:21002</connection_name>
      <application_copies>2</application_copies>
      <base_uri>
        <protocol>http</protocol>
        <machine_name>www.Company.com</machine_name>
        <virtual_directory>MyWebService</virtual_directory>
      </base_uri>
      <web_services_provider/>
    </web_config>
  </application>
</jade_config>
```

This chapter covers the following topics.

- [Overview](#)
- [JavaScripts Generated from Web Services](#)
- [Generating JavaScript](#)
- [Asynchronous Responses and Callbacks](#)
- [Client-Side Caching](#)
- [Restricted Cross-Domain Calls](#)
- [JavaScript Object Notation](#)
- [Example of JavaScript to Invoke Web Services](#)
- [Generated JavaScript API for a Web Service](#)
 - [WSUtil.js](#)
 - [Classes.js](#)
 - [Web-service_api.js](#)
 - [Web-service_types.js](#)
 - [Web-service_testharness.html](#)

Overview

The classic JADE Web application framework centered all activity on a client-server round-trip architecture with a Web service architecture. Under this system all processing is done on the server, and the client is only used to display static HTML content. The biggest drawback with this approach is that all interaction with the application must pass through the server; that is, data is sent to the server, the server responds, and a full page refresh occurs on the client with the response, when only a small part of the display requires updating.

By using a client-side technology that can execute instructions on the client's computer, Rich Internet Applications (RIAs) can circumvent this slow and synchronous loop for many user interactions and perform partial page updates.

The main characteristic of an RIA is that it has an intermediate layer of code, often called a client engine, between the user and the server. This client engine is usually downloaded at the beginning of the application, and can be supplemented by further code downloads as the application progresses. The client engine acts as an extension of the Web browser, and usually takes over responsibility for rendering the user interface and for server communication.

In a RIA, it is not uncommon for a large number of messages to be sent between a browser client and the server, but the size of each message is typically small. An example of the way this strategy works would be a user completing address and phone number details on a form in a Web browser. When the user tabs out of the phone number field, a message can be generated to validate the number that was entered. The message is sent to the appropriate JADE Web service method and the response is sent back to the browser. The browser client processes the response and takes any action that is necessary.

The JADE Rich Internet Application framework uses the JavaScript language. It is a client-side language that can run code and is installed on most Web browsers. JavaScript code is used to facilitate the messaging and event handling and the JADE Web service framework is the client-server communication link in this Rich Internet Application strategy.

The browser client, by executing JavaScript, manipulates the displayed page with incremental updates avoiding the traditional full page refreshes.

Note Although the generated JavaScript should work in most current browser versions, we recommend one of IE 6.0, IE 7.0, Firefox 2.0, Firefox 3.0, Chrome 1.0, or Safari 3.2.

JavaScripts Generated from Web Services

The JavaScript Web Service Consumer Generator is primarily designed to aid JavaScript developers invoke JADE Web services. Within the JADE development environment, the developer selects a Web service to be accessed from JavaScript and the generator creates a number of JavaScript files, which enable the JavaScript developer to invoke that Web service.

The generator supports only Web services in the **document/literal** SOAP 1.1 format.

The generated files are downloaded and used on the client. The JavaScript developer can simply interact with the messages as though they were static methods. For example, a Web service provider application named **CustomerService** provides a JADE Web service method with the following signature.

```
addCustomer(firstName: String; lastName: String; age: Integer) webService;
```

The JavaScript code to add a customer would be as follows.

```
Jade.CustomerService.addCustomer("John", "Smith", 40);
```

In this example, **Jade** is the namespace specified by the user who generated the JavaScript.

The generated API also provides the following functionality.

- Client-side cache
- Session handling
- Exception handling
- Message preprocessing
- A type-checked class framework

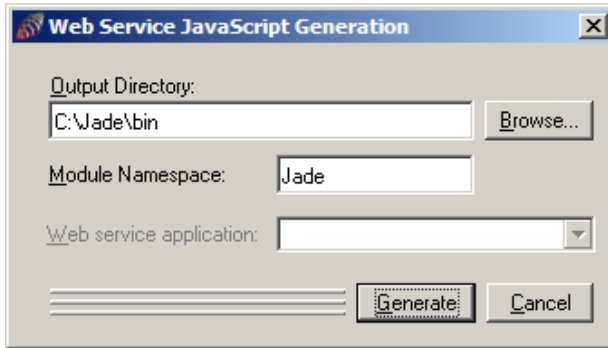
Generating JavaScript

You can generate JavaScript for a Web service provided by or consumed by JADE.

» To generate JavaScript for a Web service consumer

1. In the Web Service Consumer Browser, select the consumer subclass with which you want to interact.
2. Select the **Generate JavaScript** command from the Consumer menu.

The Web Service JavaScript Generation dialog, shown in the following example, is then displayed.



3. If you want the JavaScript files created in a directory other than the program directory, enter the location in the **Output Directory** text box. Alternatively, click the **Browse** button and the common file Open dialog is then displayed to enable you to select the appropriate location.
4. Enter a namespace in the **Module Namespace** text box. The namespace refers to the root JavaScript variable name that the generated scripts use. For example, if the namespace is **Jade**, the instruction to call a Web service from JavaScript would be in the following format.

```
Jade.<provider>.<method>(<parameters>);
```

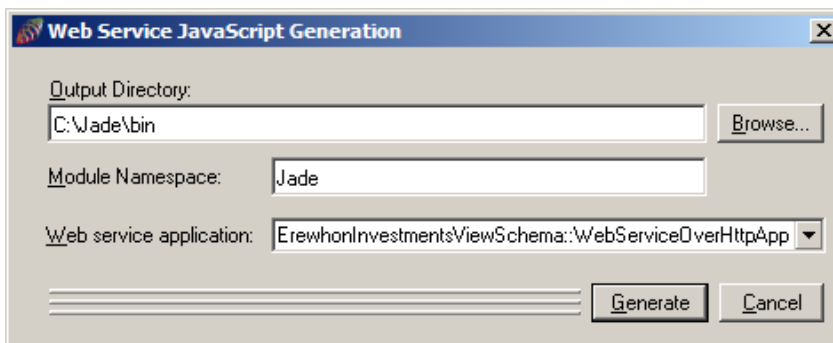
5. Click the **Generate** button. Alternatively, click the **Cancel** button to abandon your selections.

A number of JavaScript (.js) files are created in the specified directory.

» To generate JavaScript for a Web service exposure

1. In the Exposure Browser, select the Web service exposure for which you want to generate JavaScript.
2. Select the **Generate JavaScript** command from the Exposure menu.

The Web Service JavaScript Generation dialog, shown in the following example, is then displayed.



3. If you want the JavaScript files created in a directory other than the program directory, enter the location in the **Output Directory** text box. Alternatively, click the **Browse** button and the common file Open dialog is then displayed to enable you to select the appropriate location.
4. Enter a namespace in the **Module Namespace** text box. The namespace refers to the root JavaScript variable name that the generated scripts use. For example, if the namespace is **Jade**, the instruction to call a Web service from JavaScript would be in the following format.

```
Jade.<provider>.<method>(<parameters>);
```

5. Select a Web service provider application that uses the exposure in the **Web service application** combo box. This enables the generated JavaScripts to access the correct application.
6. Click the **Generate** button. Alternatively, click the **Cancel** button to abandon your selections.

A number of JavaScript (.js) files are created in the specified directory.

Asynchronous Responses and Callbacks

Within the generated framework, all responses to Web service calls are received asynchronously. Consequently, you must provide the equivalent of an event handler, called a *callback function*, to handling a response.

The following example shows the definition and registration of a callback function.

```
function handleResponse (xml) {  
    alert("The Web service replied");  
}  
  
Jade.CustomerService.getAge("John", "Smith", {callback: handleResponse});
```

The framework enables you to pass a **config** object as an additional parameter to each Web service call. You can set properties on the **config** object to specify how the framework deals with the call.

In the previous example, the **callback** property is set to refer to a function that will be invoked when the Web service replies. The function is passed the XML returned by the Web service. You could then extract the parts of the reply in which you are interested (for example, by using a helper library such as **jQuery**) and carry out processing based on the response.

Client-Side Caching

For Web service calls that repeatedly request the same, static information (for example, post codes), the RIA framework has a built-in cache to store the information. Response times for the client application can be reduced by eliminating unnecessary traffic across the Web. You can set the following options in the **config** object for the cache.

- **cache: true**

This setting causes the result of the Web service to be stored in the cache. Further calls to the same service that also have **cache: true** perform a simple cache lookup rather than invoke the Web service.

- **cacheExpiry: number-of-seconds**

This setting specifies the number of seconds the entry remains in cache before it expires. When an entry expires, the next call to the Web service method invokes the Web service rather than using the cached value.

- **cacheKey: String**

This setting is an optional secondary key for a Web service method.

The primary key is the method name, so that multiple calls to the same method use the same cached value. For a method with parameters, you would almost certainly not want to use the same cached value for all calls to that method. If a **cacheKey** value is specified, only an entry in the cache that matches the method and the **cacheKey** is considered to match the request.

Note Individual method calls with unusual parameters can bypass the cache, by not using the **cache: true** option.

Restricted Cross-Domain Calls

A security feature of modern Web browsers is the inability of JavaScript to communicate with a Web site other than its home site. For example, JavaScript at www.jadeworld.com can send messages to the server at www.jadeworld.com but not to the server at www.google.com, which has a different address.

This places severe restrictions on the use of JavaScript to invoke Web services. The service provider must reside on the same domain as the JavaScript; that is, the JADE service provider and the server on which the JavaScript is located must share the same domain name.

In a simple case, where the directory containing the `jadehttp.dll` file is the JADE binary directory `c:\jade\bin` and has the alias `jade` in Microsoft Internet Information Server (IIS), the JavaScript must reside in a subdirectory of the JADE binary directory.

The browser can access the JavaScript files using the following IIS address, as follows.

```
http://webserver/jade/Provider_testharness.html
```

Web browser security prevents direct access through the `file` URL, as follows.

```
file:///c:/Jade/bin/Provider_testharness.html
```

The accepted way to invoke cross-domain Web services from JavaScript is through your Web server; that is, arrange for the JADE Web service provider on your Web server to consume the external service (for example, the Google search service) and then create a provider in JADE to provide that service to your JavaScript.

The JavaScript performs a Web service call to the JADE service provided by your Web server, which acts as an intermediary and requests the data from Google.

JavaScript Object Notation

JavaScript Object Notation (JSON) is a way of creating an object in JavaScript with its properties initialized to specified values. A comma-separated list of `key:value` pairs is enclosed in braces, as shown in the following JSON example.

```
var person = {name:"Harry", age:17};
```

Note Do not leave a trailing comma after the last `key:value` pair in your JSON, as some Web browsers may not behave correctly.

An object is created with properties **name** and **age**, which are initialized to **"Harry"** and **17**, respectively. It is equivalent to the following JavaScript code.

```
var person = new Object();  
person.name = "Harry";  
person.age = 17;
```

JSON notation helps in making JavaScript code simpler and easier to read.

Example of JavaScript to Invoke Web Services

The following code shows the use of the framework to invoke two Web services from the Erewhon system, which is the JADE example system.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
  <title>JavaScript Web Service Consumer test</title>
  <script type="text/javascript" src="Classes.js"></script>
  <script type="text/javascript" src="WSUtil.js"></script>
  <script type="text/javascript"
    src="WebServiceOverHttpApp_types.js"></script>
  <script type="text/javascript"
    src="WebServiceOverHttpApp_api.js"></script>
  <script type="text/javascript" src="jquery.js"></script>
  <script type="text/javascript">
    var client = new Jade.WebServiceOverHttpApp_Types.Client(
      { // Construct the parameter
        name:"Barry Ogen",
        address1:"123 Straight St",
        address2:"Newtown",
        address3:"Te Anau",
        home:"031234123",
        fax:"03987654",
        email:"barry@gmail.com",
        webSite:"www.BarryOgen.com"});

    Jade.WebServiceOverHttpApp.updateClientWithProxy(client);
    // Invoke the service
    Jade.WebServiceOverHttpApp.getClient("Barry Ogen", {cache:true,
      callback:function(xml) { // invoke the service
        var email = $(xml).find("email").text();
        // use jQuery to extract information we are interested in
        alert("Barry's e-mail address is now: " + email);}});
  </script>
</head>
<body>
</body>
</html>
```

Generated JavaScript API for a Web Service

When the generation function is performed, the following files are generated.

File	Provides
Classes.js	Generic type-safe class framework
WSUtil.js	Generic essentials required to invoke Web services
Web-service_types.js	Types exposed by the Web service
Web-service_api.js	JavaScript methods to invoke the Web service methods
Web-service_testharness.html	Sample Web application used to check the framework is functioning correctly

Note There are dependencies among these files. When including them within an HTML file, include the **.js** files in the order listed in the above table.

The **WSUtil.js** and **Classes.js** files are independent of the specific Web service. If the JavaScript needs to invoke two or more separate Web services, it therefore requires only one copy of these files.

For each Web service, the corresponding **Web-service_api.js** and **Web-service_types.js** files are required.

For details about the generated JavaScript, see the following subsections.

WSUtil.js

The **WSUtil.js** file provides the following public methods and properties:

Method or Property	Description
defaultCacheExpiry: Number, or false (default value is 300)	The number of seconds after which entries in the cache expire if no cacheExpiryconfig option is set. Set this to false for infinite expiry time (that is, it never expires).
incomingSoap: String	A copy of the most-recently received message.
outgoingSoap: String	A copy of the most-recently sent message.
pre-processor: Function (default value is null)	This function is called whenever a Web services returns, prior to the callback function being invoked. The preprocessing function can return true to prevent the callback being invoked. The config option settings can override this value for individual calls.
exceptionHandler: Function, or false (default value is null)	If the service provider returns an exception, the RIA framework invokes the exception handler rather than the normal callback. The config option settings can override this value for individual calls. If no override is provided, the exceptionHandler is called. Set this to false , to ignore the exception, and null to use the default exception handler, which displays a simple message box reporting the error to the user.
convertTextToXMLDOM(String): DomNode	Parses the provided string and returns a Document Object Model (DOM) of the XML.
xmlSerialize(DomNode): String	Takes a Document Object Model of the XML and serializes it into text.
shallowCopy(Object): Object	Makes a copy of an object.
flushCache()	Clears out all cache entries.
createRequest(...)	Internal method, which is used by the RIA framework and should not be invoked directly.

Classes.js

The **Classes.js** file provides the public **Class.New()** method, which has the following signature.

```
New(classname: String, superclass: Class)
```

The **Class.New()** method creates a new JavaScript class. After the class has been created, it has a number of methods and properties that can be invoked.

As a general rule, call the **setProperty()** method after calling the **Class.New()** method, to make the class a class with *properties*, or the **isArrayOf()** method, to make an array class. After invoking one of these methods, you can create instances of the class by using the **new** keyword.

Classes with properties have *set* and *get* methods for each property. In the following example, a class is created with a **color** property. If the corresponding **color()** method is called with a parameter, the method *sets* the value of the **color** property. If it is called without a parameter, the method *gets* the value of the **color** property. The method checks the types of values used.

The following example shows the use of the **Class.New()** method to create a class with properties.

```
var Animal = Jade.Class.New("Animal");           // define the class
Animal.setProperties({color:String, age:Number}); // give it properties

var myAnimal = new Animal({color:"blue",age:14}); // instantiate it
myAnimal.age();                                // get a property - returns 14
myAnimal.color("red");                          // change a property
myAnimal.color();                              // now returns "red"
myAnimal.age(Animal);                          // throws exception - fails type-check

var Lion = Jade.Class.New("Lion", Animal); // subclass Animal
Lion.setProperties({kills:Number});         // add subclassed properties

var myLion = new Lion({kills:12,color:"yellow"}); // instantiate it
// can set properties defined in superclass
myLion.kills(45);                             // set the kills
myLion.color();                                // returns "yellow"
myLion.Classname();                            // returns "Lion"
myLion instanceof Lion;                       // returns true
myLion instanceof Animal;                     // returns true
// a subclass is an instance of its superclass
myLion.Superclass();                          // returns Animal
myLion.Properties();                          // returns {kills:Number, color:String, age:Number}
```

If the **isArrayOf()** method is called instead of the **setProperties()** method, the class is an array class. After invoking this method, you can create instances of the array by using the **new** keyword.

The following example shows the use of the **Class.New()** method to create an array.

```
var AnimalArray = Jade.Class.New("AnimalArray");
AnimalArray.isArrayOf(Animal);                // make it into an array

var myAA = new AnimalArray([myAnimal, myLion]); // instantiate it
// Can set initial contents
myAA.Array();                                 // returns [myAnimal,myLion]
myAA.Array([]);                              // sets the array to empty
myAA.Add(myAnimal);                          // adds to the array

var LionArray = Jade.Class.New("LionArray", AnimalArray);
// subclasses AnimalArray
LionArray.isArrayOf(Lion);                   // make it into an array

var myLA = new LionArray([myLion]);          // instantiate it
myLA.Add(myAnimal);                         // throws type-check exception
```

The **Class.New()** method creates a **Class** object with the following properties and methods.

Method or Property	Description
ClassName(): String	Returns the name of the class.

Method or Property	Description
isArray(): Boolean	Returns whether the class is an array class (by default they are not, unless isArrayOf(memberType) has been called).
SuperClass(): Class	Returns a reference to the superclass.
isArrayOf(memberType: Class)	Makes the class into an array class (can be called once only), with members of the specified type. Methods are added to conform to the TypedArrayClass interface.
setProperty(properties: Object)	For example, setProperty({name:String, age:Number, dob:Date, children:PersonArray}) ; gives the class new properties, and can be called once only. The names and types of the class properties are those of the properties of the parameter. Methods are added to the class to conform to the TypedPropertyClass interface.

Instances of the **Class Class** (that is, the actual objects created when you use the **new** keyword to create an instance of the class) have the following properties.

Property	Description
ClassName(): String	Returns the name of the class.
isArray(): Boolean	Returns whether the class is an array class (by default, classes are not array classes, unless isArrayOf(memberType) has been called).
SuperClass(): Class	Returns a reference to the superclass.

The **TypedArrayClass** interface provides the following additional functions.

Function	Description
MemberType(): Class	Returns the member type of the array.
<constructor>(valueArray: Array of MemberType)	Optionally invokes the Array(valueArray) function in the TypedArrayClass interface, if a parameter is provided.

Instances of the **TypedArrayClass** interface have the following additional functions.

Function	Description
MemberType(): Class	Returns the member type of the array.
Array(valueArray: Array of MemberType)	Replaces the array with the values passed, after type-checking.
Array(): Array	Returns a copy of the private internal type-checked array object.
Add(value: MemberType)	Adds a value to the array. If the value is not of MemberType , an exception is raised.

The **TypedPropertyClass** interface provides the following additional functions.

Function	Description
Properties(): Object	Returns the properties passed to the setProperty() function when the class was defined.

Function	Description
<code><constructor>(values: Object)</code>	Invokes the SetProperties(values) function on the instance, if the parameter is provided.

Instances of the **TypedPropertyClass** interface have the following additional functions:

Function	Description
<code>Properties(): Object</code>	Returns the properties passed to setProperties() when the class was defined.
<code>SetProperties(values: Object)</code>	Sets the values of any number of properties after type-checking. For example, {name: "Andrew", age: 21, children: myChildrenArray} .

Instances of the **TypedPropertyClass** interface also have *get* and *set* methods with the names specified in the **setProperties()** method, as shown in the following example if the **Person** class has **name** and **age** properties.

```
Person.setProperties({name:String, age:Number});
```

You can *set* and *get* the value of the **name** property of an instance of the **Person** class, as follows.

```
emp.name("Bill");           // sets the value of the name property
emp.name();                  // gets the value of the name property
```

Web-service_api.js

The **Web-service_api.js** file provides one method for each method defined by the Web service. Each method has parameters identical to those provided by the Web service, with an additional **config** parameter that is optional.

The **OnSessionTimeout()** method provided in the **Web-service_api.js** file enables you to specify a function to be invoked when a Web session times out. Typically, you could code a JavaScript function that asks the user to log on again. When the function is called with a **null** parameter, the callback is unset.

Note Setting an **OnSessionTimeout** callback when using a Web service that does not support session handling results in all return messages being sent to that callback function.

You can set the following properties for the **config** object.

Property	Description
<code>callback: Function</code>	This function is invoked when the Web service responds.
<code>preprocessor: Function, or false</code>	This function is invoked before the callback function. The specified function is invoked instead of any general preprocessor set in the WSUtil.js file. Setting this value to false means that no preprocessor function is invoked. The preprocessor function itself can return true , to prevent the callback being invoked.
<code>exceptionHandler: Function, or false</code>	This function is invoked instead of preprocessor and callback functions if the service provider returns an exception. Setting this value to false means that the exception is ignored and no handler function is called. If this value is not specified, the handler specified in the WSUtil.js file is used.
<code>cache: Boolean</code>	If true , checks whether the value is in the local cache before invoking the Web service. If it is not in cache, it stores the value in the cache when the service responds.

Property	Description
cacheExpiry: Number, or false	Only relevant if cache: true is set. Indicates the number of seconds the entry remains in the cache before expiring. A value of false indicates that the entry never expires. If a value is not specified, the default cache expiry setting defined in the WSUtil.js file is used. The expiry time for an item in cache is governed by the lowest value of cacheExpiry specified by the config objects of any of the calls that use it.
cacheKey: String	Provides a secondary dictionary key for entries in the cache. An entry in the cache is considered a <i>match</i> only if the method being invoked is the same and the value of cacheKey is the same.

Callbacks, Preprocessors, and Exception Handlers

For most preprocessor and callback functions, the only parameter that needs to be specified is the XML DOM tree returned by the Web service; for example:

```
function handleResponse(xml) { ... }
```

Note There is no requirement in JavaScript for the parameters of a function to match the number of parameters with which it is invoked, although a parameter cannot be used if it is not included in the signature.

The other parameters passed to these functions are:

- **InvocationArguments**
- **CacheObject**

With these additional parameters, the signature would be as follows.

```
function handleResponse(xml, InvocationArguments, cacheObject) { ... }
```

The **InvocationArguments** parameter is the parameters with which the Web service call was invoked, and is an array containing the following entries.

- The string name of the method being invoked
- An instance of one of the classes from the **Web-service_types.js** file of the Web service containing the parameters passed to the Web service method
- The **config** object, which enables the handler to determine which method call it is handling

An easier way to pass information from the caller to the handler is to declare the handler function within the scope of the caller, as shown in the following example.

```
var outerVar = "abc";
Jade.Service.GetData("mydata",{cache: true, callback: function(xml){
    alert("This is the callback function. The value of outerVar is: " +
        outerVar);
}});
```

This example uses an *anonymous* function, which is generally the easiest way to write Web service calls. In JavaScript, a function definition can be supplied where a function reference is expected. Instead of defining a function called **handleResponse** and specifying it by name as the callback function, you can simply define the function in-line without a name. Such *anonymous* functions can access all of the data that is accessible within the parent function.

If the call uses the cache, a third parameter is passed to the handler (its value is undefined if the cache is not used). The third parameter is **cacheObject**, which is the entry in the cache representing the data. You can store data that can be accessed during subsequent cache calls on this object. For example, when XML is received by the preprocessor function of a user, data can be extracted and changed into a more-useful format. The preprocessor adds a property to the **cacheObject** object, to store the data. All subsequent calls to the Web service method see the data on **cacheObject** and do not need to repeatedly transform the XML.

Note All data stored on the **cacheObject** object is lost if the cache entry expires or if the cache is flushed.

Web-service_types.js

The **Web-service_types.js** file uses the class structure provided by the **Classes.js** file to construct all the classes used by the Web service. This includes both **Array** classes and normal classes. These classes are exposed to the user and are needed to invoke Web service calls where the parameters of the method are not primitive types; for example, a **CustWeb** provider in JADE has the following Web service method.

```
addCustomers(CustomerArray) webService;
```

The **CustomerArray** class is an array of **Customer** objects. The **Customer** class has **name** and **age** properties defined. The Web service is invoked from JavaScript, using the generated framework as follows.

```
var cust1 = new Jade.CustWeb_Types.Customer(); // Create the customer
cust1.name("Harold");
cust1.age(31);
var custArr = new Jade.CustWeb_Types.CustomerArray(); // Create the array
custArr.Add(cust1);
Jade.CustWeb.addCustomers(custArr, {callback:function(){// Invoke service
    alert("Successfully added the customer");}
});
```

Web-service_testharness.html

The test harness application is an HTML and JavaScript application that behaves similarly to the standard JADE Web service test harness. For details about the standard test harness, see ["Testing the Web Services Interface"](#), in Chapter 11 of the *JADE Developer's Reference*.

You can use the JavaScript Web service test harness to test that the generated framework is working. You cannot simply double-click the generated file to load it into the browser, as the browser prevents the Web service calls because of cross-domain security restrictions. Instead, you can copy the generated files to a Web server and access them through it. For details, see ["Restricted Cross-Domain Calls"](#), earlier in this chapter.