OpenEdge® Getting Started: WebSpeed Essentials
Notices

© 2015 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Business Making Progress, Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, Deliver More Than Expected, Easy!l, Fathom, Icenium, Kendo UI, Making Software Work Together, OpenEdge, Powered by Progress, Progress, Progress Control Tower, Progress RPM, Progress Software Business Making Progress, Progress Software Developers Network, Rollbase, RulesCloud, RulesWorld, SequeLink, Sitefinity (and Design), SpeedScript, Stylus Studio, TeamPulse, Telerik, (Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. AccelEvent, AppsAlive, AppServer, BravePoint, BusinessEdge, DataDirect Spy, DataDirect SupportLink, , Future Proof, High Performance Integration, Modulus, NativeScript, OpenAccess, Pacific, ProDataSet, Progress Arcade, Progress Pacific, Progress Profiles, Progress Results, Progress RFID, Progress Progress Software, ProVision, PSE Pro, SectorAlliance, Sitefinity, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, WebClient, and Who Makes Progress are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

Please refer to the Release Notes applicable to the particular Progress product release for any third-party acknowledgements required to be provided in the documentation associated with the Progress product.

The Release Notes can be found in the OpenEdge installation directory and online at: https://community.progress.com/technicalusers/w/opendegegeneral/1329.openedge-product-documentation-overview.aspx.

For the latest documentation updates see OpenEdge Product Documentation on Progress Communities: (https://community.progress.com/technicalusers/w/opendegegeneral/1329.openedge-product-documentation-overview.aspx).

October 2015

Last updated with new content: Release 11.6.0
# Table of Contents

## Preface ........................................................................................................... 11
  Purpose ................................................................................................................... 11
  Audience .................................................................................................................. 12
  Organization ............................................................................................................ 12
  Using this manual .................................................................................................. 13
    References to ABL compiler and run-time features ............................................. 13
    References to ABL data types ............................................................................. 13
  Typographical conventions ...................................................................................... 14
  Example procedures .............................................................................................. 15
  OpenEdge messages .............................................................................................. 16
    Obtaining more information about OpenEdge messages .................................. 16

## Chapter 1: Introducing WebSpeed .............................................................. 19
  WebSpeed architecture .......................................................................................... 20
  WebSpeed components ......................................................................................... 21
    WebSpeed Workshop .......................................................................................... 21
    WebSpeed Transaction Server ......................................................................... 22
    WebSpeed Messenger ....................................................................................... 23
    NameServer ....................................................................................................... 23
    Language support .............................................................................................. 23
  WebSpeed request round-trip ............................................................................... 23
    Before the first request ...................................................................................... 24
    Web request round-trip ...................................................................................... 24
  Web programming and WebSpeed ........................................................................ 26
  WebSpeed and the OpenEdge platform ............................................................... 27
    OpenEdge Reference Architecture .................................................................... 28

## Chapter 2: Configuring WebSpeed .............................................................. 31
  WebSpeed configuration overview ....................................................................... 31
  Configuring your Web server ............................................................................... 32
    Specifying the location of static files ............................................................... 32
    Configuring virtual directories for the IIS Web server .................................... 33
    Configuring virtual directories for the Apache Web server ............................ 33
    Testing the Web server ...................................................................................... 33
    Supported Web servers .................................................................................... 34
    Supported Web browsers and preference settings ......................................... 34
  WebSpeed administration ..................................................................................... 35
Chapter 4: Running and Deploying WebSpeed Applications

Distributed WebSpeed environments

Single-machine configuration

Setting up a single-machine configuration

Startup sequence

Development network with a central WebSpeed machine

Setting up a central development machine configuration

Beyond the basic installation

Startup sequence

Development network with a dedicated Web server

Setting up a development network with a dedicated Web server configuration

Beyond the basic installation

Startup sequence

Multiple LAN development environment

Setting up a multiple LAN configuration

Beyond the basic installation

Startup sequence
Overview of sample applications.................................................................114
Running the SportsPro Sales Advisor..........................................................115
Configuring a WebSpeed broker to connect to a database..........................116

Chapter 5: WebSpeed Configuration and Management Utilities.............119
Utility command syntax..................................................................................119
WSCONFIG utility.........................................................................................120
WTBMAN utility.............................................................................................123

Index..............................................................................................................127
Preface

For details, see the following topics:

- Purpose
- Audience
- Organization
- Using this manual
- Typographical conventions
- Example procedures
- OpenEdge messages

Purpose

This manual introduces you to the WebSpeed® environment for developing Web browser-based business applications that support real-time database access and management. It explains the structure of a WebSpeed environment and how transactions are processed. The manual discusses configuring WebSpeed environments and the development tools to which you have access. Finally, the manual gives basic information on deploying WebSpeed applications and securing your WebSpeed environment.
Note:

This documentation supports the "classic" WebSpeed implementation, where a third party Web server is required to host the WebSpeed Messenger that responds to client requests.

In OpenEdge 11.6 and later releases, the Pacific Application Server (PAS) for OpenEdge is available as a host for WebSpeed applications. It integrates the functionality of a Web server, the WebSpeed Transaction server, and an AppServer in a single PAS for OpenEdge instance. As a platform for WebSpeed applications, PAS for OpenEdge has many advantages, including improvements in performance, scalability, and security.

Note that you can migrate existing WebSpeed applications to PAS for OpenEdge, or you can use the Progress Developer Studio for OpenEdge to develop new WebSpeed applications that run on PAS for OpenEdge.

See Pacific Application Server for OpenEdge: Introducing PAS for OpenEdge for more information on the advantages of using PAS for OpenEdge.

Audience

This manual is for anyone interested in learning how to create applications with the WebSpeed development environment. Knowledge of WebSpeed or SpeedScript® programming is not required. However, you should also have a working understanding of the Internet and of the World Wide Web, and some experience creating and editing HTML pages.

For more information about WebSpeed configuration, see OpenEdge Management and OpenEdge Explorer: Configuration.

Organization

Introducing WebSpeed on page 19

Introduces the WebSpeed architecture, including general information on the WebSpeed round trip process, Web programming, and the Progress® OpenEdge® platform.

Configuring WebSpeed on page 31

Provides information about configuring WebSpeed environments. The chapter also includes basic information about configuring your Web server.

Tools and ABL Support on page 53

Describes the tools and utilities used in WebSpeed application development.

Running and Deploying WebSpeed Applications on page 73

Provides information on launching WebSpeed in various environments, starting information on securing your WebSpeed environment, and how to access the WebSpeed sample applications.

WebSpeed Configuration and Management Utilities on page 119

Describes the syntax for commands and utilities documented in this manual. If this manual provides the primary documentation for a command or utility, the syntax for that command or utility appears in this appendix.
Using this manual

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is ABL (Advanced Business Language). With few exceptions, all keywords of the language appear in all UPPERCASE, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.

For the latest documentation updates see the OpenEdge Product Documentation Overview page on Progress Communities:

https://community.progress.com/technicalusers/w/openedgegeneral/

References to ABL compiler and run-time features

ABL is both a compiled and an interpreted language that executes in a run-time engine. The documentation refers to this run-time engine as the ABL Virtual Machine (AVM). When the documentation refers to ABL source code compilation, it specifies ABL or the compiler as the actor that manages compile-time features of the language. When the documentation refers to run-time behavior in an executing ABL program, it specifies the AVM as the actor that manages the specified run-time behavior in the program.

For example, these sentences refer to the ABL compiler's allowance for parameter passing and the AVM's possible response to that parameter passing at run time: "ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error." The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: "The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets."

References to ABL data types

ABL provides built-in data types, built-in class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all UPPERCASE, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.
- Wherever integer appears, this is a reference to the INTEGER or INT64 data type.
- Wherever character appears, this is a reference to the CHARACTER, LONGCHAR, or CLOB data type.
- Wherever decimal appears, this is a reference to the DECIMAL data type.
- Wherever numeric appears, this is a reference to the INTEGER, INT64, or DECIMAL data type.

References to built-in class data types appear in mixed case with initial caps, for example, Progress.Lang.Object. References to user-defined class data types appear in mixed case, as specified for a given application example.
Typographical conventions

This manual uses the following typographical and syntax conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Italic typeface indicates the title of a document, or signifies new terms.</td>
</tr>
<tr>
<td><strong>SMALL, BOLD CAPITAL LETTERS</strong></td>
<td>Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, GET and CTRL.</td>
</tr>
<tr>
<td><strong>KEY1+KEY2</strong></td>
<td>A plus sign between key names indicates a <strong>simultaneous</strong> key sequence: you press and hold down the first key while pressing the second key. For example, <strong>CTRL+X</strong>.</td>
</tr>
<tr>
<td><strong>KEY1 KEY2</strong></td>
<td>A space between key names indicates a <strong>sequential</strong> key sequence: you press and release the first key, then press another key. For example, <strong>ESCAPE H</strong>.</td>
</tr>
</tbody>
</table>

**Syntax:**

| Fixed width               | A fixed-width font is used in syntax, code examples, system output, and file names. |
| **Fixed-width italics**   | Fixed-width italics indicate variables in syntax. |
| **Fixed-width bold**      | Fixed-width bold italic indicates variables in syntax with special emphasis. |
| **UPPERCASE fixed width** | ABL keywords in syntax and code examples are almost always shown in upper case. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class. |
| **Period (.) or colon (:)** | All statements except **DO, FOR, FUNCTION, PROCEDURE, and REPEAT** end with a period. **DO, FOR, FUNCTION, PROCEDURE, and REPEAT** statements can end with either a period or a colon. |
| **[ ]**                   | Large brackets indicate the items within them are optional. |
| **[]**                    | Small brackets are part of ABL. |
| **{ }**                   | Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams. |
Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.

A vertical bar indicates a choice.

Ellipses indicate repetition: you can choose one or more of the preceding items.

### Example procedures

This manual may provide example code that illustrates syntax and concepts. You can access many of the example files, and details for installing them, from the following locations:

- A self-extracting Documentation and Samples file available on the OpenEdge download page of the Progress Software Download Center
- The OpenEdge Product Documentation Overview page on Progress Communities:


Once installed, you can locate the example files for this manual in the following path under the OpenEdge Documentation and Samples installation directory:

<table>
<thead>
<tr>
<th>This directory . . .</th>
<th>Contains examples for the following documents . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>src\prodoc\dotnetobjects</td>
<td>OpenEdge Development: GUI for .NET Programming</td>
</tr>
<tr>
<td>src\prodoc\dynamics</td>
<td>The Progress Dynamics documentation</td>
</tr>
<tr>
<td>src\prodoc\getstartoop</td>
<td>OpenEdge Development: Object-oriented Programming</td>
</tr>
<tr>
<td>src\prodoc\handbook</td>
<td>OpenEdge Getting Started: ABL Essentials</td>
</tr>
<tr>
<td>src\prodoc\interfaces</td>
<td>OpenEdge Development: Programming Interfaces</td>
</tr>
<tr>
<td>src\prodoc\json</td>
<td>OpenEdge Development: Working with JSON</td>
</tr>
<tr>
<td>src\prodoc\langref</td>
<td>OpenEdge Development: ABL Reference</td>
</tr>
<tr>
<td>src\prodoc\prodatasets</td>
<td>OpenEdge Development: ProDataSets</td>
</tr>
<tr>
<td>src\prodoc\tranman</td>
<td>OpenEdge Development: Translation Manager</td>
</tr>
<tr>
<td>src\prodoc\visualdesigner</td>
<td>OpenEdge Getting Started: Introducing Progress Developer Studio for OpenEdge Visual Designer</td>
</tr>
</tbody>
</table>
OpenEdge messages

OpenEdge displays several types of messages to inform you of routine and unusual occurrences:

• **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.

• **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.

• **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

After displaying a message, OpenEdge proceeds in one of several ways:

• Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.

• Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.

• Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.

• Terminates the current session.

OpenEdge messages end with a message number in parentheses. In this example, the message number is 200:

```
** Unknown table name table. (200)
```

If you encounter an error that terminates OpenEdge, note the message number before restarting.

Obtaining more information about OpenEdge messages

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

• Choose **Help > Recent Messages** to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.

• Choose **Help > Messages** and then type the message number to display a description of a specific OpenEdge message.

• In the Procedure Editor, press the HELP key or F1.
On UNIX platforms, use the OpenEdge pro command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

To use the pro command to obtain a message description by message number:

1. Start the Procedure Editor:

   ```bash
   OpenEdge-install-dir/bin/pro
   ```

2. Press F3 to access the menu bar, then choose Help > Messages.
3. Type the message number and press ENTER. Details about that message number appear.
4. Press F4 to close the message, press F3 to access the Procedure Editor menu, and choose File > Exit.
Introducing WebSpeed

WebSpeed® is an ABL (Advanced Business Language) development and deployment environment.

Note:
This documentation supports the "classic" WebSpeed implementation, where a third party Web server is required to host the WebSpeed Messenger that responds to client requests.

In OpenEdge 11.6 and later releases, the Pacific Application Server (PAS) for OpenEdge is available as a host for WebSpeed applications. It integrates the functionality of a Web server, the WebSpeed Transaction server, and an AppServer in a single PAS for OpenEdge instance. As a platform for WebSpeed applications, PAS for OpenEdge has many advantages, including improvements in performance, scalability, and security.

Note that you can migrate existing WebSpeed applications to PAS for OpenEdge, or you can use the Progress Developer Studio for OpenEdge to develop new WebSpeed applications that run on PAS for OpenEdge.

See Pacific Application Server for OpenEdge: Introducing PAS for OpenEdge for more information on the advantages of using PAS for OpenEdge.

WebSpeed allows you to build applications that use HTML, XML, WML, DHTML, and most other mark-up languages (MLs) as the user interface. This means that WebSpeed can be used for applications where users are accessing the application using:

- A Web browser (HTML, DHTML, or XML).
- A mobile/cell-phone (HTML, WML).
- A system making requests for information using XML and HTTP or HTTP/S as the transport protocol.
In addition to building Web-based applications by design, you can use WebSpeed to Web-enable existing OpenEdge applications that previously ran stand-alone or as an OpenEdge AppServer™-based application.

With WebSpeed, you can develop and deploy:

- Intranet applications that allow internal users to access and modify data.
- Internet applications that allow external, consumer access (for example, a shopping cart application).
- Extranet, business-to-business applications.

For details, see the following topics:

- WebSpeed architecture
- WebSpeed request round-trip
- Web programming and WebSpeed
- WebSpeed and the OpenEdge platform

**WebSpeed architecture**

The WebSpeed environment is similar to the OpenEdge AppServer™ environment. A transaction server, which consists of brokers and agents, execute requests from a client. The unique piece of the WebSpeed environment, the WebSpeed Messenger, is a process that runs on your Web server capturing and redirecting client requests. The following figure illustrates the complete architecture for a WebSpeed deployment environment.
The dashed arrows in the previous figure indicate connections that do not occur in all WebSpeed configurations. The WebSpeed agents might not have direct access to a database or a DataServer. Depending on how you architect your application, the procedure that the agent runs in response to a Web request might call another procedure over an AppServer to process database requests. The Progress NameServer might not be used in all configurations, as described in the NameServer on page 23.

WebSpeed components

The components of the WebSpeed environment are the WebSpeed Workshop, the WebSpeed Messengers, and the WebSpeed Transaction Server. The WebSpeed environment can also include a NameServer, which can support both AppServer and WebSpeed transactions.

A default WebSpeed installation provides one predefined WebSpeed broker and one predefined NameServer. You can use these predefined components as templates from which you create and configure additional instances of the WebSpeed broker and, if needed, the NameServer.

WebSpeed Workshop

The WebSpeed WorkShop contains the tools that you use to develop and test WebSpeed applications. The default WebSpeed Workshop installation also includes a version of the WebSpeed Transaction Server scaled to support a single developer's activities. The Workshop includes the following:
AppBuilder — The AppBuilder is a multi-purpose application development environment that supports a broad, integrated range of application and development options. You can use it as a visual programming environment to create character- or GUI-based client/server applications. In addition, you can use the AppBuilder for WebSpeed to create HTML-based Web applications. The AppBuilder only runs in Windows platforms. You can configure it to work with a WebSpeed Transaction Server installed on a separate UNIX machine.

WebTools — You use the browser-based WebTools to access information on your server, such as the status of CGI Variables. You can also access database information, use the WebSpeed File tools, and access virtual system table data. You can use the Scripting Lab to write and test WebSpeed code, such as HTML that includes Embedded SpeedScript, and send operating system commands. With the Editor WebTool, you can create, open, save, and print files; check syntax; and compile code.

PRO*Tools — PRO*Tools is a set of utility programs that are useful for developing and running OpenEdge applications. For example, one of the PRO*Tools allows you to edit your PROPATH. The Color Changer, Screen Scaling Utility, and ProtoGen PRO*Tools do not apply to WebSpeed.

WebSpeed Transaction Server

The WebSpeed Transaction Server consists of the processes that handle the server-side activity of your WebSpeed applications:

WebSpeed agent — An application process that can execute Web objects, perform database transactions, and dynamically merge data into HTML format. The agent is the standard character ABL client running in batch mode. An AppServer agent is a single AVM instance running on the AppServer.

Note: The agent process is inherently stateless. This means that the agent is only busy when a request is being processed. It will be idle at all other times.

WebSpeed broker — An application that can do the following:

- Register with a NameServer the application services that it provides to fulfill requests from HTML clients.

  For information on running WebSpeed from a client other than the HTML client (for example, an ActiveX page or a Java application), see the Progress Communities WebSpeed Forum: http://communities.progress.com/pcom/community/psdn/openedge/webspeed

- Manage connections between clients and a pool of WebSpeed Agents.
- Maintain the status of each agent in its pool and dynamically scale the number of agents according to changing demand.

Note: If you start a WebSpeed broker without specifying a username, the Broker inherits the account that the AdminServer is using. This is generally the system account, which might not have access to network drives.
WebSpeed Messenger

The WebSpeed Messenger listens for WebSpeed requests coming in to the Web server. The Messenger asks the NameServer where to send each request. Alternately, the Messenger can bypass the NameServer as described in the NameServer on page 23. The Messenger then handles the transfer of data between the Web server and the WebSpeed Agent. There are Messengers for use on different Web servers: a CGI Messenger, an ISAPI Messenger, and an NSAPI Messenger.

There is also a Messenger that works with Microsoft's Active Server Pages, the WSASP Messenger. Using the WSASP Messenger, you can call out of an Active Server Page to a WebSpeed application. The WebSpeed Messenger always resides on the same machine with your Web server. Because the Messenger is not itself an OpenEdge application, it is sometimes the only part of the WebSpeed environment installed on a Web server machine. This is sometimes incorrectly described as a "Messenger-only deployment." Your WebSpeed applications cannot run without a WebSpeed Transaction Server. "Messenger-only installation" is a more appropriate term for this setup.

NameServer

The NameServer is a basic part of the OpenEdge architecture. It maintains a list of available AppServers and WebSpeed Transaction Servers. Those servers register the application services that they provide with the NameServer. The NameServer can then direct client connection requests to a broker that supports a requested application service. This provides scalability and location transparency to your applications.

The NameServer can also provide load balancing and fault tolerance for OpenEdge server applications. Load balancing allows you to balance client workload among multiple brokers that support the same application service (that is, the same set of procedures and resources). This ability makes the NameServer very useful in deployed applications that handle large volumes of requests.

The NameServer works through the UDP network protocol. For various reasons, some network administrators might not want UDP on their networks. To accommodate this preference, the OpenEdge architecture includes a "No NameServer" connection procedure. If you employ the "No NameServer" connection procedure in a WebSpeed application, you must configure the WebSpeed Messenger to point directly to a specific WebSpeed broker. This approach can limit the scalability of your application. For more information, see Configuring WebSpeed on page 31.

Language support

The WebSpeed development environment also includes a programming language, SpeedScript, and a number of pre-coded conveniences, such as global variables, preprocessors, and APIs, to simplify your development. For more information on these elements, see Tools and ABL Support on page 53.

WebSpeed request round-trip

Before you begin architecting a WebSpeed application, you should learn how information passes between the components when they process a request. How information passes between the components impacts such considerations as which components are installed together, how you manage session context, and how you apply security.
Before the first request

Before the first Web request is processed, the components have to start and pass some basic information. In general, this process runs as follows:

1. Start an AdminServer.
2. Start a database in multi-user mode with properties supplied by the AdminServer from the `conmgr.properties` file.
3. Start a NameServer with properties supplied by the AdminServer from the `ubroker.properties` file.
4. Start a WebSpeed broker with properties supplied by the AdminServer from the `ubroker.properties` file.
5. The broker spawns its WebSpeed agents using the information it received from the AdminServer. By default, the broker sends a message every 30 seconds to notify the NameServer that it is still available to accept requests. If the NameServer does not get a message, it deletes the broker from its “available” list. These messages are not part of the request process.
6. The broker registers itself and the application services that it provides with the NameServer.
7. Start a Web server, which makes its WebSpeed Messenger available to transfer Web requests.

Web request round-trip

The common administration mechanism provided by the AdminServer lets you spread the WebSpeed components across your network. This flexibility gives you choices in how to set up your WebSpeed configuration. In the configuration shown in the following figure, the Web server and the WebSpeed Messenger are on the same machine, as required. The AdminServer, the NameServer, and the WebSpeed Transaction Server (consisting of a WebSpeed broker and its WebSpeed agents) reside on a second machine.
The numbered steps of a Web request in the previous figure are explained in the following sequence:

1. The HTML client, running in a Web browser, generates a connection request. The request is in the form of a URL and is sent to a Web server, which forwards it to a WebSpeed Messenger.

2. The Messenger sends a request to the NameServer for an available WebSpeed broker that supports the required application service.

   Note: In a "No NameServer" configuration, the Messenger is hard-wired with connection information for a single WebSpeed broker. The Messenger passes all Web requests directly to that broker.

3. The NameServer selects a broker, which supports the requested application service, from the pool of brokers that have registered with it. The NameServer sends the broker's host name or IP address and the broker's port number to the Messenger.

4. Using these details, the Messenger connects to the broker and requests a WebSpeed agent to process the request. This request is put in a queue by the broker, so requests are not lost in peak load times.

5. If there are requests in the queue, the broker checks for an available agent in its pool. The broker allocates the next available agent to the request and marks that agent as busy. The broker then returns that agent's port number to the Messenger.

   Note: If there are no free agents and the broker's maximum number of agents has not been reached, the broker starts a new agent to process the request.

6. The Messenger connects to the agent through that port and passes the Web request to the agent.

7. The agent executes the Web request and creates an HTML page that it returns to the Messenger.
8. The agent informs the broker that it is available again.

9. The Messenger passes the HTML page to the Web server, which passes it back to the HTML client.

Step 2 on page 25 through Step 5 on page 25 create only small amounts of network traffic, usually less than 500 bytes. The large amounts of data are in the final request and response, Step 6 on page 25 and Step 7 on page 25. The data sent from the Messenger to the WebSpeed agent includes all of the environment variables, as well as the input parameters from the URL or HTML form. The environment variables alone can be up to 3000 bytes. When the response comes back from a WebSpeed agent, it could be a simple HTML page of around 1000 bytes, but it also could be a large .ZIP file or similar. With special programming, WebSpeed can send binary files to the Web browser.

All of these components (the Web server, the WebSpeed Messenger, the WebSpeed broker, the WebSpeed agents, and the NameServer) can reside on a single physical machine. However, you can also distribute them on separate machines, with the following restrictions:

- The Web server and the WebSpeed Messenger must reside on the same physical machine.
- The NameServer can reside on any machine, but requires an AdminServer.
- The WebSpeed Transaction Server (the WebSpeed broker and the WebSpeed agents) requires an AdminServer on its machine. The broker and the agents that it supports must reside on the same physical machine. You can have multiple WebSpeed Transaction Servers spread over several machines, but registered with a single NameServer.

See Configuring WebSpeed on page 31 for examples of how to distribute the components across a network. You can also distribute the databases over a network. For more information about distributing OpenEdge databases across a network, see OpenEdge Data Management: Database Administration. For information on accessing a non-OpenEdge data source, see the appropriate OpenEdge DataServer guide. When you read these manuals, which describe a typical client/server environment, substitute the term Agent for Client.

Web programming and WebSpeed

WebSpeed is used to manipulate, customize, and automate facilities for Web-based applications. It allows you to develop and deploy Internet-based applications that use XML, HTML, DHTML, WML, and Java by embedding SpeedScript directly into your HTML pages, or by using HTML mapping to bind HTML files to business logic.

WebSpeed can be deployed in environments leveraging:

- **Cascading Style Sheets (CSS)** — Provides a simple mechanism for adding style characteristics to Web documents. For more information, refer to http://www.w3.org/Style/CSS.

- **Extensible Markup Language (XML)** — XML is a simple and flexible text format derived from SGML. It was originally designed to meet the challenges of large scale electronic publishing, but it is also playing an important role in the exchange of a wide variety of data on the Web. For more information, refer to http://www.w3.org/XML.

- **Wireless Markup Language (WML)** — WML inherits traits based on HTML and XML and is used to run simple code on the client. For more information, refer to http://www.w3.org.

- **Hypertext Markup Language (HTML)** — HTML is the standard language for publishing hypertext on the Web. It is a nonproprietary format based on SGML, and can be used to process a wide range of tools.
• **Dynamic Hypertext Markup Language (DHTML)** — DHTML allows you to control the display and positioning of HTML elements in the browser. This language is a combination of HTML, CSS, and JavaScript.

Web-based applications developed using WebSpeed are run in a Web browser. A Web browser provides the host environment of client-side computation, including objects representing windows, menus, pop-ups, dialog boxes, text areas, anchors, frames, history, cookies, and input/output functionality. In addition, the Web browser provides a means to attach scripting code to events such as a change of focus, page and image loading, unloading, error and abort, selection, form submission, and mouse actions. WebSpeed coding appears within the HTML, and the displayed page is a combination of user interface elements and fixed and computed text and images.

For information on supported browsers, see Supported Web browsers and preference settings on page 34.

### WebSpeed and the OpenEdge platform

As a part of OpenEdge, WebSpeed applications can connect to the rest of the OpenEdge application server platform. For instance, a WebSpeed application can call other OpenEdge applications across an AppServer. You should become familiar with the other parts of the OpenEdge platform, as well as WebSpeed. For a basic look at all the pieces of the application server picture, see *OpenEdge Getting Started: Application and Integration Services*.

At their core, WebSpeed applications are ABL applications. Generally speaking, most things you can do with the ABL you can do with WebSpeed. However, coding your entire application as a WebSpeed application is not necessarily the best practice. When you plan to deploy an application on multiple clients, you should modularize the code so that only tasks that differ between clients are duplicated. Business logic that is not client-specific should be shared by all clients. The more modular your code is, the easier it is for you to maintain and reuse the code. Modularization is just one practice that makes your initial development efficient and eases later efforts to adapt to changing technology and business needs.

### OpenEdge Reference Architecture

The OpenEdge Reference Architecture is a recommended approach to designing business applications according to current best practices. The reference architecture views an application as a set of layers that provide services to each other, as shown in the following figure. This model allows your business and support logic to be modularized for flexibility and reusability. You can use the reference architecture as a whole, or adopt it a piece at a time to fit your needs. The reference architecture separates business tasks into a set of layers.
The procedures in the Data Access layer manage handling data from your data stores. These procedures retrieve information from wherever it resides in the physical data stores and arrange the data into logical datasets that meet the business needs for the procedures in the Business Servicing layer. Other procedures in the Data Access layer extract the data changes from the logical datasets and commit the changes to the proper places in the physical data stores.

The procedures in the Business Servicing layer act on requests received from users through the Presentation layer or from enterprise services through the Integration layer. These procedures handle the business tasks required to fulfill an order, for example. The procedures in the Business Servicing also push data changes in the data sets back to the Data Access layer.

The procedures in the Presentation and Integration layers pass requests from external sources (users or enterprise services) to the Business Servicing layer. Procedures in these layers might prevalidate that user requests are complete and in the proper format. The main work of these layers is to transform incoming data into the form needed by the business logic of the Business Servicing layer and to properly present the results for the consumers.

This is only a brief sketch of the OpenEdge Reference Architecture. For more information, see Progress Communities: Architecture, SaaS & Cloud Computing Community at http://communities.progress.com/pcom/community/psdn/openedge/architecture.

**WebSpeed and the OpenEdge Reference Architecture**

If you adopt the OE Reference Architecture, you must consider where WebSpeed applications are appropriate choices. Before coding a task into the WebSpeed part of your application, you should consider if it really belongs there. What is the essential difference between a WebSpeed application and an equivalent ABL application?
A WebSpeed application and an equivalent ABL application use the same data to complete the same business task. So, they would use the same modules in the Data Access layer. To perform the same task, both applications would use the same business logic. So, they would use the same modules in the Business Servicing layer.

The essential difference between WebSpeed and ABL applications is how they gather information from and present results to the user. A WebSpeed application uses an HTML client (or a client based on some other markup language). This point positions WebSpeed applications as elements of the OE Reference Architecture’s Presentation layer, as shown in the following figure.

**Figure 4: WebSpeed's use in OpenEdge Reference Architecture**

In an application built according to the reference architecture, a WebSpeed component passes user requests to the appropriate procedures in the Business Servicing layer and passes the results back to the user. This role limits the kinds of tasks that you would code into WebSpeed procedures.

For example, the following tasks are generally appropriate for the Presentation layer:

- Validating that the fields in a form are filled in with appropriate values.
- User interface control tasks, such as populating a secondary combo box based on the selection in the primary combo box.

The following tasks are generally not appropriate for the Presentation layer:

- Calculation routines, such as figuring price totals or sales tax.
- Direct database access for anything other than a UI control task.
Configuring WebSpeed

This chapter provides configuration information for the WebSpeed environment, including aspects of your Web server.

For more information about using OpenEdge Management or OpenEdge Explorer to configure WebSpeed, see OpenEdge Management and OpenEdge Explorer: Configuration located on Progress Communities https://community.progress.com/technicalusers/w/openedgegeneral/1329.openedge-product-documentation-overview.aspx or the OpenEdge Management or OpenEdge Explorer online help.

For details, see the following topics:

- WebSpeed configuration overview
- Configuring your Web server
- WebSpeed administration
- Setting up the WebSpeed environment
- Maintaining the WebSpeed Transaction Server and NameServer log files
- Configuring a WebSpeed Transaction Server
- Configuring a WebSpeed Messenger

WebSpeed configuration overview

You must perform the following preliminary tasks before you can begin configuring WebSpeed:
• Install a Web server and verify connectivity.

• Install the necessary WebSpeed components. You can distribute WebSpeed components over a number of machines, but the WebSpeed Messenger must be installed in the scripts directory of your Web server.

• Configure the machines where WebSpeed components are installed. This includes setting the appropriate environment variables and setting up your Web server.

For more information on installing WebSpeed, see *OpenEdge Getting Started: Installation and Configuration*.

## Configuring your Web server

WebSpeed uses a Web server to host the WebSpeed Messenger and the static HTML, images, and other files used by the Web browser to render the HTML. This section provides general guidelines about both tasks. It also provides specific information about configuring the Microsoft IIS and the Apache Web servers. If you are running a Web server other than IIS or Apache, you must refer to the documentation for that Web server for configuration information.

### Specifying the location of static files

Static files include HTML, JavaScript, Cascading Style Sheet, application, and graphics files. The Web server can find these files through the use of virtual directories, or it can find them when they are installed under the Web Server’s Document Root directory. You choose where to put the static files during the installation of WebSpeed, either copying them to the Document Root Directory or creating a virtual path to them. The actual location of the Document Root Directory depends on your Web server.

Rather than physically moving static files into or below the Document Root, you can set up a virtual directory that points to the directory (*install_dir*/tty/webtools) that contains the static files. For more information on setting up virtual directories, see Configuring virtual directories for the IIS Web server on page 33 or Configuring virtual directories for the Apache Web server on page 33. Refer to individual product documentation for other Web servers.

The static files for Web applications reside under a relative path structure with main module files separated into subdirectories, as shown in the following table.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Link pages for testing and tools (index.html, default.htm)</td>
</tr>
<tr>
<td>path\dhtml</td>
<td>DHTML files: .css, .htm, and .js</td>
</tr>
<tr>
<td>path\img</td>
<td>Images</td>
</tr>
</tbody>
</table>

**Note:** In the table, path can be either a virtual directory that points to *install_dir*/tty/webtools, or a document_root_dir\ (where document_root_dir is the Document Root directory of your Web server).
Caution: These files are only used for the WebSpeed Workshop. For security reasons, they should not be available in nondevelopment environments.

Configuring virtual directories for the IIS Web server

You create a virtual directory for WebSpeed static files so that the Web server can find them. (The other alternative is to copy the static files to the Web server's Document Root directory.) You can think of a virtual directory as an alias for install_dir\tt\webtools, the directory where the WebSpeed install places the static files.

You can use the IIS Web server’s Internet Service Manager to set up virtual directories. The Internet Service Manager can usually be found under the Administrative Tools section of the Windows Control Panel. When it is running, select Default Web Site. Then select Action > New > Virtual Directories. A wizard takes you through the process of creating a virtual directory.

Note: These directories should not be marked as "executable."

Configuring virtual directories for the Apache Web server

You create a virtual directory for WebSpeed static files so that the Web server can find them. (The other alternative is to copy the static files to the Web server's Document Root directory.) You can think of a virtual directory as an alias for install_dir\webspeed, the directory where the Progress Dynamics install places the static files.

For the Apache Web server, edit (apache_install_dir/conf/httpd.conf) and create a webspeed alias for OpenEdge_install_dir/webspeed. The following example shows the entry:

```
Alias /webspeed/ "OpenEdge_install_dir/webspeed/

<Directory "OpenEdge_install_dir/webspeed">
   Options Indexes MultiViews
   AllowOverride None
   Order allow,deny
   Allow from all
</Directory>
```

Note: The Apache Web Server must be restarted after you make configuration changes.

Testing the Web server

Before you can test the Web server, you must have at least one WebSpeed broker. In addition, make sure that the Web server is running, and that the NameServer and AdminService is properly configured.

To verify that your Web server has been set up correctly:
1. Verify that the Web server is running.
2. Start a Web browser. (Internet Explorer Version 6.0 or later is recommended.)
3. Use ping functionality to determine if connectivity exists between the Web server and WebSpeed components. To accomplish this, conduct a round-trip test from the browser to a WebSpeed agent using the CGI Messenger (cgiip.exe). This test instructs the Messenger to make a connection to the broker and an available agent.

An example of ping for a Web server in Windows follows:

```
```

An example of ping for a Web server on Linux or UNIX follows:

```
```

Note: The host name (identified as localhost in the example) differs depending upon the environment in which WebSpeed operates; your installation choice (for example, on a local machine) dictates the path.

**Supported Web servers**

For WebSpeed application development, you can use any Web server that supports one of the following interfaces:

- **CGI 1.1** — For example, Microsoft Internet Information Server (IIS), Apache, Netscape Enterprise, or Fast Track Server
- **ISAPI** — For example, Microsoft Internet Information Server (IIS)
- **NSAPI** — For example, Netscape Enterprise or Fast Track Server

**Supported Web browsers and preference settings**

The WebSpeed development environment requires Netscape Navigator (Version 4.5 or later) or Microsoft Internet Explorer (Version 6.0 or later).

The following table lists the recommended Web browser settings for the WebSpeed Workshop environment.

<table>
<thead>
<tr>
<th>Preference</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk Cache</td>
<td>5000K</td>
</tr>
<tr>
<td>Memory Cache</td>
<td>1000K</td>
</tr>
<tr>
<td>Number of Connections</td>
<td>4</td>
</tr>
</tbody>
</table>
WebSpeed administration

WebSpeed administration consists of the following:

- The AdminService, which provides access to OpenEdge servers to be administered on the local machine
- The ubroker.properties file, which dictates property values for the WebSpeed Transaction Server, WebSpeed Messengers, and the NameServer
- OpenEdge Management or OpenEdge Explorer, which allow local and remote administration and configuration of WebSpeed and other OpenEdge components
- The management utilities, which allow administration from the command line of WebSpeed and other OpenEdge components
- The optional NameServer

This framework provides a consistent structure for all the OpenEdge server products installed on your network.

AdminService

The AdminService supports the managing of WebSpeed and other OpenEdge products (for example, NameServer, database, DataServer).

The AdminService runs as a service on UNIX and Windows platforms. In Windows, it starts automatically by default. To start the AdminService on UNIX or Linux, use the proadsv utility.

To start the AdminService if you have altered the default behavior:

1. From the Windows taskbar, choose Start > Control Panel > Administrative Tools > Services.
2. Select the AdminService for OpenEdge, and click Start.

Alternately, you can run a command from a command prompt or a batch file similar to the following:

```
NET START "AdminService for OpenEdge version"
```

Where version is the version number of OpenEdge. You can find the version number for your installation by going to the OpenEdge folder in your Windows Start menu and choosing Version Info.
ubroker.properties file

The ubroker.properties file is the property file for the WebSpeed Transaction Server, WebSpeed Messengers, and the NameServer. All values that define instances of the WebSpeed Transaction Server and the NameServer are stored within this file. The command-line utilities and OpenEdge Management/OpenEdge Explorer access this information through the AdminServer when working with instances of all processes.

The ubroker.properties file resides in the OpenEdge-install-dir/properties directory. It is a fully commented file containing information relevant to setting properties for your WebSpeed configuration.

Note: The AppServer and the DataServers also use the ubroker.properties file to store configuration data. For the purposes of this guide, the ubroker.properties file focus is on the WebSpeed Transaction Server and the NameServer. See the appropriate manual for details about viewing and editing configurations applicable to the other products.

From a Windows or a UNIX machine, you use OpenEdge Management or OpenEdge Explorer to create and configure instances of the WebSpeed Transaction Server or the NameServer on the Windows platform or remote UNIX platforms. It is possible to edit the ubroker.properties file manually. See Editing the ubroker.properties file on page 36 for more information. Advanced users can also use the mergeprop utility to apply changes to the ubroker.properties file. For more information on the mergeprop utility, see the chapter on managing OpenEdge property files in OpenEdge Getting Started: Installation and Configuration.

The ubroker.properties file consists of a hierarchical structure of configuration entities, where parent entities provide configuration information that you can override or extend in each child entity. Each configuration entity has a name that begins the entity definition, and the definition contains configuration settings for one or more product instances. When configuring your WebSpeed environment, you work most often with the [UBroker], [UBroker.WS], [NameServer], [WebSpeed], and [WebSpeed.Messengers] configuration entities.

Editing the ubroker.properties file

You can edit ubroker.properties directly using any text editor to create new WebSpeed Transaction Server and NameServer configurations or edit existing configurations. The simplest way to make new configurations in the ubroker.properties file is to copy an existing Transaction Server or NameServer definition and then modify the values of the copy’s properties to suit your needs. When you do this, you must be sure to supply each definition with its own uuid setting, as described in the list of required unique parameters later in this section.

From a Windows or a UNIX machine, you can also use OpenEdge Management or OpenEdge Explorer remotely to create and configure instances of the WebSpeed Transaction Server or the NameServer on the UNIX platform.

If you instead edit the configuration using a text editor, note that:

- The ubroker.properties file resides in the /properties subdirectory of the OpenEdge installation directory.
- You should not directly change the values in the ubroker.properties file unless you have a complete understanding of how the changes affect WebSpeed components. If you have OpenEdge Management or OpenEdge Explorer available from a remote Windows machine, use it to make all changes to this file on your UNIX machines.
• For complete definitions of all the properties and detailed information about how to set them, see the OpenEdge-install-dir\properties\ubroker.properties.README file.

• If you create additional instances of the WebSpeed Transaction Server and the NameServer, you must be sure that each of the following parameters has a value unique to the entire ubroker.properties file:
  - [Ubroker.WS.broker-name] — The Transaction Server name must be unique.
  - portNumber — Each Transaction Server configuration requires a unique port number.
  - defaultService — You can only set one default service on each NameServer. If you configure two WebSpeed Transaction Servers to use the same NameServer and specify that the Transaction Servers perform the same application service, the Transaction Servers must also support the same business function.
  - appserviceNameList — A list of aliases that can be used in the WService=<appservice-name> in the URL.
  - uuid — A universally unique identifier for a Transaction Server. If you use OpenEdge Management or OpenEdge Explorer to create the new Transaction Server, this property is automatically set. If you manually add Transaction Server definitions, generate a unique uuid for each Transaction Server definition by using the following command:

    ```
    install-path\bin\genuuid
    ```

    You can then enter that value in the file.
  
  - Log files — Each Transaction Server configuration and NameServer configuration generates its own log files; the names must be unique for each Transaction Server and each NameServer.

• If you create additional instances of the WebSpeed Transaction Server and the NameServer by copying an existing instance, be sure that each of the following parameters has the correct values for the new instance:
  - srvrStartupParam — Identify the startup parameters for your agents. Copy the value from the ubroker.properties file's [UBroker.WS] section to your new Transaction Server definition, and modify.
  - controllingNameServer — Make sure that each Transaction Server points to its controlling NameServer.
  - userName and groupName — You can optionally specify a username and a group name that the Transaction Server runs under; if you do not specify these names, the Transaction Server runs under the username and group name of the user who starts the AdminServer.

**Note:** If you install the NameServer on a separate host from the WebSpeed Transaction Server, the NameServer installation includes its own copy of the properties file. You also must configure WebSpeed to use a remote NameServer. You must ensure that all related properties and sections of the file are properly specified for each Transaction Server or NameServer instance. If you do edit the file directly, use the appropriate configuration utility (NSCONFIG or WSCONFIG) to validate the product configuration that you have edited. For more information on utilities, see WebSpeed command-line utilities on page 38 and WSCONFIG utility on page 120. in OpenEdge Application Server: Administration.
Unified Broker framework

The Unified Broker framework combines the functionality of all the command-line utilities with the ability to create, save modifications to, and delete individual WebSpeed Transaction Servers, NameServers, DataServers, AppServers, SonicMQ Adapters, and databases. You can also use the Unified Broker framework to configure WebSpeed Messengers, start additional WebSpeed agents, or trim back running WebSpeed agents.

The Unified Broker framework also includes OpenEdge Management/OpenEdge Explorer, either of which is a preferred method for performing most tasks involving the framework.

When you install WebSpeed, a sample WebSpeed Transaction Server (wsbroker1) and a sample NameServer (NS1) are installed automatically. You can administer each product instance through OpenEdge Management or OpenEdge Explorer. In addition, you can configure or modify all of the properties for a specific instance. You can also start, save, delete, check the status of, or stop a WebSpeed broker or NameServer instance.

For more information, see the documentation on Progress Communities (https://community.progress.com/technicalusers/w/openedgegeneral/1329.openedge-product-documentation-overview.aspx) or the OpenEdge Management or OpenEdge Explorer online help.

WebSpeed command-line utilities

While OpenEdge Management and OpenEdge Explorer are the preferred tools for configuring and administering WebSpeed Transaction Servers, there are also command-line utilities available. The available utilities are:

- **WTBMAN** — Use the WTBMAN utility to control the operation of a WebSpeed Transaction Server. The utility allows you to start a Transaction Server, query its status, start and stop additional WebSpeed agents, trim by a certain number of agents, and shut down the Transaction Server.

- **WSCONFIG** — The WCONFIG utility validates existing WebSpeed Transaction Server or WebSpeed Messenger configurations. The WCONFIG utility reads the ubroker.properties file for validation.

  The WCONFIG configuration command runs locally only, on the machine where the WebSpeed components that you want to check are installed.

**Note:** Because the WCONFIG utility does not run across the network and no AdminService is installed during a Messenger-only installation, you cannot use the WCONFIG utility to check a Messenger-only installation.

For more information, see WebSpeed Configuration and Management Utilities on page 119.

NameServer

The NameServer serves as a hub through which a WebSpeed Messenger can locate a WebSpeed Transaction Server that provides the application services needed to fulfill a Web request. The NameServer provides location transparency that can ease deployment of your applications. The Enterprise version of the NameServer can also supply load balancing. Load balancing can improve performance and provide fault tolerance.
Understanding the NameServer’s load balancing option

Load balancing is a feature that allows client connection requests to be distributed, based on load, among multiple Unified broker instances that support the same Application Service. If you have installed the load-balancing option, the NameServer assigns client connections to the appropriate Unified broker instances based on weight factors that you specify.

If the weight factor that you specify for each Unified broker instance is appropriate in relation to the others, the effect is to assign more connections to broker instances with greater resources, and thus to balance connection load among all the instances. You can set the load-balancing weight factor for each Unified broker instance in OpenEdge Management/OpenEdge Explorer or by editing the `priorityWeight` property in the `ubroker.properties` file.

Percentage weight factors

Properly specified, these weight factors give some sense of the amount of work that an individual WebSpeed Transaction Server instance can handle. For example, the following table shows the effect of weight factors specified for three WebSpeed Transaction Server instances registered for the same application service.

Table 3: Weight factors based on percentage

<table>
<thead>
<tr>
<th>WebSpeed Transaction Server name</th>
<th>Weight factor</th>
<th>% of time selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>WS2</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>WS3</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>

The selection algorithm used by the NameServer guarantees that WS1 and WS2 are each selected 20% of the time and WS3 is selected 60% of the time. Thus, if the sum of weight factors for all WebSpeed Transaction Server instances that support the same application adds up to 100, each weight factor specifies the exact percentage of time that the NameServer selects the given WebSpeed Transaction Server instance over time.

Arbitrary weight factors

You can specify any sum of values (not necessarily 100), but the weight of each is always proportional to the sum, as shown in the following table.

Table 4: Weight factors based on arbitrary sums

<table>
<thead>
<tr>
<th>WebSpeed Transaction Server name</th>
<th>Weight factor</th>
<th>% of time selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS1</td>
<td>2</td>
<td>$2/7 = 28.57$</td>
</tr>
<tr>
<td>WS2</td>
<td>2</td>
<td>$2/7 = 28.57$</td>
</tr>
<tr>
<td>WS3</td>
<td>3</td>
<td>$3/7 = 42.86$</td>
</tr>
</tbody>
</table>
Fail-over weight factor

Another use of Weight Factors is to provide connection-level fault tolerance by keeping a backup Transaction Server ready to take over the load if the other Transaction Servers in your network fail. If you assign a Weight Factor of zero to the backup Transaction Server, the NameServer does not select it to fill any of the incoming Web requests. If the NameServer loses connection with all the other Transaction Servers that support the application service, the NameServer directs all the Web requests to the backup Transaction Server.

For more information on load balancing and fault tolerance, see OpenEdge Getting Started: Installation and Configuration.

"No NameServer" configurations

While the NameServer can be useful, use of the NameServer is optional. There are several reasons why you might want to eliminate the NameServer from your configuration:

- Simple configurations or low-demand deployments might not require location transparency and load balancing.
- The NameServer uses the User Datagram Protocol (UDP). Some sites have restrictions that prohibit the use of UDP.

If you choose not to use the NameServer, configure your Transaction Server to indicate that it should not register with a NameServer. Then, configure your Messenger to connect directly to the Transaction Server. For more information, see Configuring a WebSpeed Transaction Server on page 45 and Configuring a WebSpeed Messenger on page 48. You can also see OpenEdge Management and OpenEdge Explorer: Configuration on Progress Communities (https://community.progress.com/technicalusers/w/openedgegeneral/1329.openedge-product-documentation-overview.aspx) or the OpenEdge Management or OpenEdge Explorer online help.

You can also eliminate the NameServer by directly editing the ubroker.properties file, although using OpenEdge Management or OpenEdge Explorer is less error-prone. For more information on the ubroker.properties file, see ubroker.properties file on page 36.

To eliminate the NameServer by editing the ubroker.properties file:

1. Open OpenEdge-install-dir/properties/ubroker.properties in a text editor.
2. Find the broker definition for your Transaction Server. For example:

   [UBroker.WS.wsbroker1]

   Note that wsbroker1 is the name of the Transaction Server.

3. Add and set the following property:

   registerNameServer=0

   Or, if registerNameServer is already set to 1, reset it to 0.
4. Find the definition for your Messenger. For example, if you use CGIIP:

```
[WebSpeed.Messengers.CGIIP]
```

5. Add and set the following property:

```
registerNameServer=0
```

Or, if registerNameServer is already set to 1, reset it to 0.

6. Add and set the port number for your broker. For example, if you are using the default wsbroker1:

```
Port=3055
```

7. Save and close the ubroker.properties file.

**Note:** When you eliminate the NameServer, the Messenger can access only one WebSpeed Transaction Server (broker). One of the advantages of using the NameServer is that you can run multiple brokers.

---

**Setting up the WebSpeed environment**

Once you complete the WebSpeed installation, you must configure it to your particular requirements. You can manage the configuration of WebSpeed on the Windows and UNIX platforms by using OpenEdge Management/OpenEdge Explorer or the WebSpeed command-line utilities, described earlier in this chapter.

The following sections describe how to set up WebSpeed on the machine that will run the broker and agents. Remember that your Web server machine and the machine that runs the broker and agents can be the same machine. However, you might want to use more than one machine to run WebSpeed, depending on the size of your WebSpeed application and the computing power of your Web server machine.

**Creating a working application directory**

You must create a working directory for your WebSpeed application. Usually, this directory is where you place all of your WebSpeed application files—your .r, .w, .off, and .htm files. This directory also serves as the working directory for your broker and agents (this is the recommended configuration, but is not required). The workDir parameter, which you set in the ubroker.properties file or with OpenEdge Management/OpenEdge Explorer, tells the broker which directory to use as its working directory. This is also the working directory for all of the agents in the broker’s agent pool.
Moving application files to appropriate directories

After you create a working directory, move the appropriate application files into it. However, you must move some files to an appropriate Web server directory, such as the document root. This section describes the common files that you must place in appropriate directories.

Web objects and procedures

Whenever you create an executable procedure using the AppBuilder, you create a Web object. Sometimes you create Web objects that are mapped to HTML files, but this is not required to dynamically generate HTML pages. Either way, during development, you must move all of your WebSpeed procedure files (.w, .p, or .i) into your application PROPATH. This ensures that the agents can find and execute your WebSpeed procedures. If you precompile all of your procedures (creating .r files), you can remove the source files when you actually deploy the application. For more information, see Compiling Web objects on page 43.

HTML and other static files

Some of the HTML files and images that you use in your Web presentation might never be processed by WebSpeed. These HTML files serve as static HTML pages that never change throughout the life of your Web presentation. You must place these files on your Web server machine so that your Web server can serve them directly to the Web user. However, if your WebSpeed application uses HTML files to dynamically generate HTML pages, you must make the HTML files visible to the broker and agents. The easiest way to achieve this is to place them in your application working directory (the broker's default directory) or elsewhere in the PROPATH.

Java class files

You should place Java class files in a subdirectory of the Web server root directory. Then you can reference the subdirectory in the CODEBASE attribute of the APPLET tag and the Java class file in the CODE attribute of the APPLET tag.

For example, if the Java class file, myclass.class, is in a subdirectory of the Web server root directory called java_classes, the APPLET tag might look like the following:

```
<APPLET CODE="myclass.class" CODEBASE="/java_classes"></APPLET>
```
JavaScript files

You should place JavaScript (.js) files in a subdirectory of the Web server root directory. Then you can reference the relative path in the SRC attribute of the `<SCRIPT>` tag.

For example, if the JavaScript file, `myscript.js`, is in a subdirectory of the Web server root directory called `javascript`, the `<SCRIPT>` tag might look like the following:

```
<SCRIPT LANGUAGE="JavaScript" SRC="/javascript/myscript.js"></SCRIPT>
```

tagmap.dat

If you modify the default `tagmap.dat`, place a copy of the modified `tagmap.dat` into your working directory. If you do not modify the file, you do not have to copy the file because the default `tagmap.dat` is used.

Offset files

An offset file (.off) is created whenever you use the AppBuilder to map an HTML file to a Web object. Agents use the offset file information to dynamically generate an HTML page. The purpose of the offset file is to provide the location of the HTML form fields in the HTML file. You ensure that the agents can find your offset files by placing them in your working directory or in the directory with your running Web objects (or r-code). If an offset file is not current for its HTML-mapping Web object, the agent generates a new offset file from your mapped HTML file and the available `tagmap.dat` file.

Note: Some of the files described above are ASCII files and some are binary. Some transfer methods automatically handle the differences. Other methods require that you specify a type. When in doubt, specify binary.

Compiling Web objects

Typically, you compile Web objects as you develop them in the AppBuilder. For example, when you save a source file in a Procedure Window, a dialog box appears asking if you want to compile the file.

There might be occasions when you want to compile multiple files. You can compile single or multiple files in the WebTools File Tools interface. From the AppBuilder main menu, select Tools > WebTools. Then select File Tools from the left frame of the browser. The File Tools interface has a list box, from which you can select multiple files, and a compile button.

Setting environment variables

Environment variables required by WebSpeed are set during installation in the `ubroker.properties` file. You might need to set the `PROPATH` and other standard environment variables (for example, `DLC`) on the WebSpeed Transaction Server machine.
**Note:** In distributed configurations, you must edit the appropriate environment variables on each machine where you have WebSpeed components installed.

You can change most of these settings using OpenEdge Management/OpenEdge Explorer or by editing the WebSpeed property file, ubroker.properties. Note that it is not necessary to modify the Windows registry or the system environment variables (through the Windows Control Panel).

### PROPATH and other standard OpenEdge environment variables

When you install the WebSpeed Transaction Server, the installation process sets the PROPATH for you in the ubroker.properties file. The PROPATH initially includes a number of subdirectories in your installation directory. In addition, the PROPATH includes a dot (.) directory reference.

When the agent sees the dot, the process substitutes the name of its current working directory. For example, the agents resolve the dot to their broker’s default directory, which is the working directory.

You can override installed PROPATH settings using the PROPATH property in the properties file (ubroker.properties).

### Working directory settings

The properties file relies on a default setting for the working directory that you specify during installation. You can remove or modify the references in the properties file to establish your own working directory settings for both the WebSpeed Transaction Server and the NameServer.

For more information on OpenEdge environment settings, see *OpenEdge Getting Started: Installation and Configuration*.

### Configuring WebSpeed and NameServer log files

For disk management reasons, you might want to specify a non-default location for the log files used by WebSpeed. A WebSpeed installation uses a number of different log files, which are stored in the default working directory. For example:

- **[TransactionServername.]server.log** — WebSpeed Transaction Server log file set using the srvrLogFile property in the [Ubroker] section of the properties file
- **[TransactionServername.]broker.log** — WebSpeed broker log file set using the brokerLogFile property in the [Ubroker] section of the properties file
- **[NameServername.]ns.log** — NameServer log file set using the srvrLogFile property in the [NameServer] section of the properties file

After you decide where you want the log files to reside, you can specify the location for each in OpenEdge Management/OpenEdge Explorer or by directly editing the ubroker.properties file. For more information, see *WebSpeed administration* on page 35.

Because the log files receive the WebSpeed and NameServer startup and shutdown messages, OpenEdge system messages, and trace messages, the files can grow quickly. If you have the Append option set in the Transaction Server's configuration, these log files do not truncate automatically. In this case, you should periodically trim the files with a text editor. You might want to archive the contents of the files as you do it. For more information on maintaining log files, see *Maintaining the WebSpeed Transaction Server and NameServer log files*. 
Maintaining the WebSpeed Transaction Server and NameServer log files

The WebSpeed and NameServer log files include information on when the respective WebSpeed and NameServer processes start up and shut down. These files also include system messages and information up to a level of detail that you specify using the `loggingLevel` property in the `ubroker.properties` file. In addition, the WebSpeed broker and agent log files include all OpenEdge system messages that are not assigned an output destination by the `OUTPUT TO KEEP MESSAGES` statement. If you start a WebSpeed session with the `--weblogerror` startup parameter, messages from your ABL (Advanced Business Language) code can be captured in the agent log file. Thus, the information in these log files can be useful when you perform routine maintenance or troubleshooting.

**Note:** If you have the Append option set in the Transaction Server's configuration, these log files do not truncate automatically. In this case, you should periodically trim the file with a text editor. You might want to archive the file contents as you do it.

For more information on how to configure the log files for your environment, see Configuring WebSpeed and NameServer log files on page 44.

Configuring a WebSpeed Transaction Server

The WebSpeed Transaction Server consists of the processes that handle the server-side activity of your WebSpeed applications. Use the following procedure to configure the WebSpeed Transaction Server and NameServer for developing and deploying WebSpeed applications in Windows.

**Note:** For more information about procedures for configuring the WebSpeed Transaction Server, see OpenEdge Management and OpenEdge Explorer: Configuration or the OpenEdge Management or OpenEdge Explorer online help.

To prepare your WebSpeed environment for the transaction server:

1. Make sure the AdminService is running. If the AdminService is not running, you must start it. (For information on starting it, see AdminService on page 35.)
2. Start an existing NameServer or create a new NameServer instance. You can create and start a NameServer by using OpenEdge Management/OpenEdge Explorer or you can edit the `ubroker.properties` file to create an instance and then use the NSMAN utility to start the instance. When you configure a NameServer instance, you can set it to start up by default whenever the AdminService starts.

**Note:** The NameServer can be on any machine in your network, even a UNIX machine. You can configure a WebSpeed environment without a NameServer. For more information, see "No NameServer" configurations on page 40.

If you are using OpenEdge Management or OpenEdge Explorer, see the online help for information about creating and starting an instance. If you are editing the `ubroker.properties` file, see Editing the `ubroker.properties` file on page 36.
To start a local instance of the NameServer from the command line, use the following command:

```bash	nsman -name NS1 -start
```

Where **NS1** is the name of the NameServer.

To start a remote instance of the NameServer from the command line, use the following command:

```bash	nsman -name NS1 -host host-name -port port -user user-name -start
```

Where **host-name** is the name of the host machine on which you want the instance to run, **port** is the port number on the AdminService, and **user-name** is the user ID of the system account that started the AdminServer.

3. Start an existing WebSpeed Transaction Server or create a new Transaction Server instance. You can create and start a Transaction Server by using OpenEdge Management/OpenEdge Explorer or you can edit the `ubroker.properties` file to create an instance and then use the `WTBMAN` utility to start the instance. When you configure a Transaction Server instance, you can set it to start up by default whenever the AdminService starts.

To start a local instance of the WebSpeed Transaction Server from the command line, use the following command:

```bash
wtbman -name wsbroker1 -start
```

**Note:** The WebSpeed Transaction Server consists of a broker and agents. The default WebSpeed broker is `wsbroker1`. When you start the broker, the agents are also started.

To start a remote instance of the WebSpeed Transaction Server from the command line, use the following command:

```bash
wtbman -name broker -host host -port port -user user -start
```

Where **broker** is the name of the WebSpeed broker, **host** is the name of the host machine on which you want the instance to run, **port** is the port number on the AdminService, and **user** is the user ID of the system account under which the Transaction Server will run. If you specify a host name, the tool prompts you for a user name (if you do not supply it) and password.

By using either the tools or the command-line utilities, you can also stop a NameServer or WebSpeed Transaction Server instance, check its status, and increase or reduce the number of running WebSpeed agents. For more information, see the OpenEdge Management and OpenEdge Explorer online help and Managing the WebSpeed Transaction Server on page 47.
Managing the WebSpeed Transaction Server

The WebSpeed Transaction Server is a background process and, as such, does not present a visual interface to the administrator. However, you can manage the Transaction Server with the WTBMAN utility, and you can validate its properties in the ubroker.properties file with the WSCONFIG utility. The WTBMAN utility commands can run remotely; the WSCONFIG utility commands only run locally.

In addition to allowing you to start, query, or stop the Transaction Server, the WTBMAN utility lets you do the following:

- Start additional agents
- Trim the number of agents
- Stop the Transaction Server
- Display command-line help

Dynamically starting additional agents

To start additional agents, enter the following command:

```
wtbman -name broker -addagents number-to-start
```

Where `broker` is the name of the WebSpeed broker specified in the ubroker.properties file and `number-to-start` is the number of additional agents you want to start. The number you specify must not exceed the `maxSrvInstance` value in the ubroker.properties file or your license limit.

Trimming running agents

To trim agents, enter the following command:

```
wtbman -name broker -trimagents number-to-trim
```

Where `broker` is the name of the Transaction Server and `number-to-trim` is the number of agents you want to stop.

Stopping the WebSpeed broker

To stop the broker and all the agents in its pool, enter the following command:

```
wtbman -name broker -stop
```
To force an immediate shutdown of the Transaction Server and all its agents, enter the following command:

```
wtbman -kill broker
```

### Accessing help on WTBMAN

To get information on WTBMAN syntax and usage, enter the following command:

```
wtbman -help
```

---

## Configuring a WebSpeed Messenger

The WebSpeed Transaction Server and WebSpeed Messenger combine to allow an application written in SpeedScript® (the WebSpeed variant of the ABL) to use a Web browser as its user interface. Essentially, the WebSpeed Transaction Server becomes an application server for Web browser clients. The Transaction Server runs the SpeedScript, which generates HTML pages, and returns these Web pages to Web browser clients through the Messenger in response to Post and Get requests from the clients.

The Messenger runs on a Web server as a CGI or equivalent process (depending on the Web server type) and acts as the gateway and translator between Web requests and responses on the Web server side and the corresponding WebSpeed requests and responses on the Transaction Server side. Like the AppServer, the WebSpeed Transaction Server can have a controlling NameServer configured to provide server-level fault tolerance with multiple Transaction Servers supporting the same application service. The WebSpeed Transaction Server and Messenger each rely on the Unified Broker framework for configuration and administration.

**Note:** There is also a Messenger that works with Microsoft's Active Server Pages, the WSASP Messenger. Using the WSASP Messenger, you can call out of an Active Server Page to a WebSpeed application.

For information on developing SpeedScript applications for WebSpeed deployment, see *OpenEdge Application Server: Developing WebSpeed Applications*.

---

### Configuring a Messenger-only installation

WebSpeed supports n-tier deployment, enabling flexible network structuring so that you can distribute application logic and processing load among many machines across your distributed network. You can partition and deploy your application, thereby breaking up an application that is large and flexibly reorganizing it to run in a network environment that better suits your needs. The WebSpeed Messenger can be part of this n-tier deployment.

The WebSpeed Messenger must reside on the same machine as the Web server. The Web server and the WebSpeed Messengers need not be on the same machine as the rest of the WebSpeed components.
In this configuration, the Messenger must be able to connect remotely to the machines where the NameServers and AdminServices are installed. To do this, you must configure a remote NameServer. For more information on this, see *OpenEdge Getting Started: Installation and Configuration*.

## Installing the Messenger executable

The WebSpeed Messenger must reside on the same machine as your Web server. The Messenger executables are tailored to run with a specific type of Web server. WebSpeed provides a Messenger executable that supports the Web server types described in .

### Table 5: Windows NT WebSpeed Messengers

<table>
<thead>
<tr>
<th>Web server Type</th>
<th>Messenger</th>
<th>Messenger executable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft IIS</td>
<td>ISAPI</td>
<td>wsisa.dll</td>
</tr>
<tr>
<td>Microsoft IIS</td>
<td>WSASP(^1)</td>
<td>wsasp.dll</td>
</tr>
<tr>
<td>Netscape</td>
<td>NSAPI</td>
<td>wnsnsa.dll</td>
</tr>
<tr>
<td>CGI-compatible</td>
<td>CGI</td>
<td>cgiip.exe</td>
</tr>
</tbody>
</table>

The NSAPI executables reside and run from the `install-path\bin` directory. The CGI Messenger and ISAPI executables reside and run from the `\scripts` directory on the Web server.

**Note:** To configure a Netscape Web server to work with the WebSpeed NSAPI Messenger, you must edit the Netscape Web server configuration file (`obj.conf`). For more information, refer to *Editing the Netscape Web server configuration file* on page 50.

You can use the sample file `cgiip.wsc` to set up a file association for running the CGIIP Messenger under Microsoft’s IIS Server. For details, see the `cgiip.wsc` file, which is located in the `install-path\bin` directory.

**Note:** You must restart an ISAPI or Netscape NSAPI Web server after installing and configuring the Messenger.

### Compatibility with the Web server type and operating system

The Messenger executable comes with the WebSpeed Transaction Server. The Messenger is installed in the `scripts` directory of the Web server, which you specify during an OpenEdge installation.

\(^1\) The WSASP Messenger calls WebSpeed applications from an Active Server Page. It cannot coexist with the ISAPI Messenger.
If you install the WebSpeed Transaction Server on an operating system that is different from the operating system that runs your Web server, go to the Download area on the PSDN Web site at http://www.progress.com/esd/index.ssp and download a compatible Messenger. This is necessary because you must install a Messenger that is compatible with the Web server’s operating system. For example, if you are distributing WebSpeed components across networked machines and your Web server is running on UNIX, be sure to install a WebSpeed Messenger executable that is suitable for running on UNIX. Make sure that you download a Messenger for the appropriate UNIX platform and for the appropriate Web server type.

Editing the Netscape Web server configuration file

A Netscape Web server uses information in its configuration file to recognize the WebSpeed NSAPI Messenger. The configuration file for the Netscape Enterprise Server is named \install-dir\https-host-name\config\obj.conf. (If you are using the Fast Track Server, see your Web server documentation for the name of the server's configuration file.)

Make a copy of the file before you modify it so that you can restore the original configuration. The following table describes the changes you must make to obj.conf.

Table 6: Netscape Web server configuration

<table>
<thead>
<tr>
<th>New Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init fn=load-modules \nshlib=&quot;pathname&quot; \nfuncs=WSNSAinit,WSNSAdefault, \nWSNSAshutdown,WSNSAwebspeedCheck</td>
<td>This pathname is the absolute path to the Messenger DLL, wsnsa.dll. The Init line must appear after any existing Init commands. This line informs the Web server that it must load the Messenger DLL at the end of the initialization process, and that the named functions are external entry points within the DLL.</td>
</tr>
<tr>
<td>Init fn=WSNSAinit</td>
<td>This Init line must appear as the last Init command. It informs the Web server that the named function is an external entry point within the DLL.</td>
</tr>
<tr>
<td>NameTrans fn=WSNSAwebspeedCheck</td>
<td>This NameTrans line must appear before any existing NameTrans commands. You must add it within the block of commands delimited by &lt;Object name=default&gt; and &lt;/Object&gt;. This line informs the Web server to handle WebSpeed requests properly.</td>
</tr>
<tr>
<td>Service method=(GET</td>
<td>POST</td>
</tr>
</tbody>
</table>
Each line you add to obj.conf must be on a single line. Do not add line breaks within a command line. Use forward slashes (/) in pathnames. Here is an excerpt from a sample obj.conf file (the additions that you must make for the WebSpeed Messenger are bold):

```
Init ...
Init ...
# The following directive is a single line; it contains no line breaks
Init fn=load-modules shlib="c:/Program Files/OpenEdge/bin/wansa.dll"
  func=WSNSAninit,WSNSAdefualt,WSNSAshutdown,WSNSAwebspeedCheck
Init fn=WSNSAinit
<Object name=default>
  AuthTrans ...
  AuthTrans ...
  NameTrans fn=WSNSAwebspeedCheck
  NameTrans ...
  NameTrans ...
  PathCheck ...
  PathCheck ...
  ObjectType ...
  ObjectType ...
  Service method=(GET|POST|HEAD) fn=WSNSAdefualt
  Service ...
  Service ...
  AddLog ...
  AddLog ...
  Error ...
</Object> ...
...
...
```

**Applying the configuration changes**

Access the Netscape Server's browser-based Admin panel and apply the configuration changes before restarting the Web server.

**Restarting the Netscape Web server**

After editing the Netscape Web server configuration file (obj.conf) to support the WebSpeed NSAPI messenger, you must restart the Web server so that it recognizes this newly installed Messenger.

Before running any WebSpeed application, make sure that your Web server is up and running. Consult your Web server documentation for more information about getting the Web server fully up and running.

**Where to place the Messenger executable file**

Typically, a Messenger script file, such as cgiip.exe, resides in the \scripts or equivalent directory that contains your Web server's scripts. When you configure your Web server, you can decide which directories can hold executable files.

Most Web servers map URLs leading with /scripts to a /scripts subdirectory. This subdirectory is located either under or parallel to the document root directory. The /scripts directory typically contains only executable files. This is an appropriate location to place your Messenger script file.
Managing the WebSpeed Messenger

Occasionally, you might want to check the Messenger's status. For example, the NSAPI Messenger maintains persistent connections to the WebSpeed brokers, which you might want to break manually. WebSpeed provides an Administrative HTML page for the Messenger that you can access through a URL.

If you are running an NSAPI Web server, use the following URL:

```
http://host-name[:port]/wsnsa.dll[/WService=appservice-name]?WSMAdmin
```

Where `host-name` is the name of the host on which the Messenger is running, `port` is the port that your Web server uses (if different from the default port 80), and `appservice-name` is the name of the application service.

For example, the following URL requests the Administration page for the NSAPI Messenger on a host named `mars`:

```
http://mars/wsnsa.dll/WService=wsbroker1?WSMAdmin
```

If you are running an ISAPI Web server, use the following URL:

```
http://host-name[:port]/scripts/wsisa.dll[/WService=appservice-name]?WSMAdmin
```

If you are running a CGI Web server, use the following URL:

```
http://host-name/scripts/cgiip.exe[/WService=appservice-name]?WSMAdmin
```

In a UNIX environment, use the following URL:

```
http://host-name/cgi-bin/wspd cgi.sh?WSMAdmin
```

Where `host-name` is the name of your Web server machine, `port` is the port that your Web server uses (if different from the default port 80), `scripts` is your Web server's scripts directory, and `appservice-name` is the name of the application service.
Tools and ABL Support

This chapter introduces you to the tools and utilities used in WebSpeed application development. There are Windows-based and browser-based tools. You access the Windows-based tools through the AppBuilder. Browser-based tools, also known as WebTools, can be launched by starting a browser from the AppBuilder, or you can run them by supplying a URL directly in a Web browser. For details, see the following topics:

- **AppBuilder**
- **Progress Developer Studio for OpenEdge**
- **WebSpeed Error Customization Utility**
- **WebTools**
- **Language support**

AppBuilder

The AppBuilder is a multi-purpose application development environment. You can use it as a visual programming environment to create character- or GUI-based client/server applications. In addition, you can use the AppBuilder to create WebSpeed applications.

The AppBuilder is installed as part of the OpenEdge® Studio and WebSpeed Workshop products.

This manual focuses on using the AppBuilder in the context of WebSpeed applications. For more information about using the AppBuilder for other types of applications, see *OpenEdge Development: AppBuilder*. 

---

OpenEdge Getting Started: WebSpeed Essentials 53
Visual and nonvisual objects

When you start the AppBuilder, a separate window called the Object Palette appears. The Object Palette allows you to create visual objects (buttons, for example).

However, in WebSpeed, you only create nonvisual objects, which include embedded SpeedScript, CGI Wrapper, and HTML-mapping Web objects. (WebSpeed SpeedScript is an interpreted, block-structured, and statement-oriented language based on the ABL (Advanced Business Language.)

Note: You cannot manipulate nonvisual objects graphically at design time, and they do not have a visualization element at run time.

WebSpeed wizards

You can use the WebSpeed wizards to create Web objects without writing any HTML or SpeedScript code. The WebSpeed wizards prompt you for the necessary information and automatically generate the required code.

Note: WebSpeed wizards do not conform to modern coding practices and might not be appropriate to your development objectives. They are useful for quickly generating small, stand-alone applications.

The WebSpeed wizards are:

- **Report** — Creates a tabular view of the database fields you specify. You can add navigation buttons for displaying the first, previous, next, and last set of records. You can also include a text entry field to allow the user to enter search criteria.

- **Detail** — Creates a form to display database data, based on the selection criteria you specify. You can add transaction control buttons to allow users to add or delete records and to submit or reset changes.

- **HTML mapping** — Maps form fields in an existing HTML file to database fields.

Templates

The AppBuilder provides templates to assist you when you create HTML files. From the main window of the AppBuilder, choose **File > New** to see the list of templates that are available.

The following WebSpeed templates are available:

- **Detail** — Allows you to create an HTML Detail Page with Embedded SpeedScript. It creates an updatable form to display a single record from a Progress SmartDataObject™ or from a database. Optionally, you can add transaction control buttons to allow users to add, submit, reset, delete and cancel their updates. You can also add navigation buttons and an entry field that allows the user to enter search criteria.

- **HTML Mapping** — Permits you to create a new HTML Mapping procedure and associate form elements defined in a static HTML file with WebSpeed field objects, such as database or data object fields.

- **Main** — A Welcome or a static text page for a WebSpeed application.
• **CGI wrapper** — A Web object that dynamically generates HTML within the context of SpeedScript.

• **Blank** — A basic HTML file that includes an embedded SpeedScript section.

• **Frameset** — An HTML file that creates three frames (a banner and two columns). The template also contains markup that displays a message when the browser does not support frames.

• **Report** — An HTML file with preprocessor definitions for creating a tabular view of database fields with navigation buttons.

• **Report Template** — Generates a formatted report on database or SmartData tables. The user can customize the reporting options in this template by changing Preprocessor definitions to match the data to include in the report.

• **Table** — A file for formatting database results into an HTML table. This is not a stand-alone file since it does not contain all the required HTML tags. It is meant to be inserted into another HTML file.

**Code Section Editor**

You can edit procedure files with conventional text editors, such as NotePad, or with the Progress Procedure Editor. However, the Code Section Editor is the AppBuilder’s main tool for editing structured procedure files. The Code Section Editor is integrated with the AppBuilder, unlike other editors. As a result, when you invoke the Code Section Editor, it displays the appropriate code section, preserving the structure of structured procedure files. The Code Section Editor also preserves the procedure file’s format by presenting only some sections of the file.

To invoke the Code Section Editor, choose the **Edit code** button or **Window > Code Section Editor** from the AppBuilder main menu. You can also double-click on an object to invoke the Code Section Editor, if that feature is enabled in the **Options > Preferences** window.

**Starting the AppBuilder**

You can start the AppBuilder from the **Start** menu, typically by choosing: **Start > All Programs > OpenEdge > AppBuilder**. The main window of the AppBuilder opens, as shown in the following figure.

**Figure 5: AppBuilder main window**

In addition, you can start the AppBuilder from the Application Development Environment (ADE) Desktop and any tool that has a **Tools** menu, like the Data Dictionary or the Procedure Editor. You can also start the AppBuilder directly from an operating system command shell.

For more information about starting the AppBuilder and about the AppBuilder main window, see *OpenEdge Development: AppBuilder*. 
Connecting to a database server

You must connect AppBuilder to a database server before you can begin to develop WebSpeed applications. The database must be either the same database that is connected to the WebSpeed broker, or it must have the same schema.

To connect to a database server that is running:

1. From the AppBuilder main window, choose Tools > Database Connections.
2. Choose Connect.
3. Choose Options>>.
4. Select the Multiple Users check box.
5. Type the pathname of the database in the Physical Name text field.
6. Click OK.

For more detailed information about starting database servers, see OpenEdge Getting Started: Installation and Configuration.

Specifying a default browser

To run and test your WebSpeed applications, you must specify a default browser.

To specify a default browser:

1. Choose Options > Preferences from the AppBuilder main window.
2. Choose the WebSpeed tab.
3. Specify the pathname of the browser that you intend to use during development.

For example, C:\Program Files\Netscape\Communicator\Program\netscape.exe and C:\Program Files\Internet Explorer\iexplore.exe are typical pathnames for Netscape Navigator and Internet Explorer, respectively.

Specifying a WebSpeed broker

Before you can begin to develop WebSpeed applications, you must specify a WebSpeed broker in the AppBuilder. The WebSpeed broker must be connected to the same database that is connected to the AppBuilder (or the database must have the same schema as the database that is connected to the AppBuilder).

Running sample applications on page 114 contains an example of starting a WebSpeed broker connected to the sample Sports2000 database.

To specify a WebSpeed broker in the AppBuilder:

1. Choose Options > Preferences from the AppBuilder main window.
2. Choose the WebSpeed tab.
3. Type the URL of the WebSpeed broker in the Broker URL field.

The following code shows the general syntax for specifying a broker URL:

```
http://host_name[:port]/scripts_dir/messenger/WService=broker
```
**host_name**

Specifies the name of the machine that is running the Web server.

**port**

Specifies the port number of the Web Server. The port number is optional if the Web Server uses the default port number, which is 80. For example, if a Web Server's port number is 88, the initial part of the URL might be specified as `http://myhost:88`.

**scripts_dir**

Specifies the Web server scripts directory for a CGI or ISAPI Messenger. Omit this component if you are using an NSAPI Messenger.

**messenger**

Specifies the messenger name.

For a CGI Messenger, add the filename (or pathname relative to the scripts directory) of your CGI Messenger script (cgiip.exe). For an ISAPI Messenger, add the filename (or pathname relative to the scripts directory) of the DLL (`wsisa.dll`). For an NSAPI Messenger, add the filename of the DLL (`wsnsa.dll`).

**broker**

Specifies the WebSpeed broker name.

4. Click **Test** to verify the connection. (Be sure that you specify the path of your default browser’s executable file before testing.)

5. Click **OK**.

## Setting the remote/local development mode

The AppBuilder supports a local and a remote development mode.

Use local mode when all of the following conditions are true:

- The AppBuilder is running on the same machine as the WebSpeed Transaction Server
- Both the AppBuilder and the WebSpeed Transaction Server have the same working directory
- Both the AppBuilder and the WebSpeed Transaction Server have the same **PROPATH** settings

The working directory and **PROPATH** are set during installation of WebSpeed. For more information, see *OpenEdge Getting Started: Installation and Configuration*.

Use remote mode when the WebSpeed Transaction Server is on a remote machine, and when either of the following conditions are true:

- The WebSpeed Transaction Server and the AppBuilder have different **PROPATH** settings.
- The WebSpeed Transaction Server has a different working directory than the AppBuilder. (In remote mode, files are saved in the WebSpeed Transaction Server's working directory on the remote machine.)

To change the development mode, click the **Development Mode** toggle button, which is the last button in the AppBuilder tool bar as shown in the following figure.
**Figure 6: Development Mode button**

![Development Mode button](image)

**Note:** If you have network or other problems and cannot access a remote WebSpeed Transaction Server, you can switch to local mode to create and save source files. However, you cannot compile source files that contain embedded SpeedScript unless remote mode is enabled.

**AppBuilder documentation**

Detailed online help for all the AppBuilder features is available from the main Help menu and from Help buttons on most dialog boxes. Also, see OpenEdge Application Server: Developing AppServer Applications for more information about using the AppBuilder.

**Progress Developer Studio for OpenEdge**

Progress Developer Studio for OpenEdge is an Eclipse-based OpenEdge product that supports creating and managing WebSpeed applications. Features include:

- A WebSpeed project type and a workspace that hosts the tools, editors and views that support the development and deployment of WebSpeed applications
- Templates and wizards for creating SpeedScript and CGI Wrapper files
- Full-featured editors for editing HTML, Embedded SpeedScript, and CGI Wrapper files
- Support for publishing WebSpeed applications to local and remote servers
- Support for running, testing, and debugging WebSpeed applications on local and remote servers
- Support for configuring and managing local and remote servers

In many respects, OpenEdge Developer Studio is an excellent alternative to the AppBuilder for WebSpeed application development. The ability to instantly publish your code to a variety of servers is a great advantage that is not available in other tools. However, OpenEdge Developer Studio does not support the HTML Mapping coding model.
WebSpeed Error Customization Utility

The WebSpeed Error Customization Utility allows you to change the content of any OpenEdge error message that is passed by the WebSpeed Messenger to client browsers. For example, you can shield the users of your application from seeing error numbers and direct them to contact a system administrator instead.

In general, the WebSpeed Messenger passes default error messages to client browsers when WebSpeed requests generate errors. Using the WebSpeed Error Customization Utility, you can:

• Retain the default error message
• Replace the default error message with a message that you create
• Pass a URL to the client browser, which directs the client browser to a specified Web site

To access the WebSpeed Error Customization Utility, you must enable the Administration Utility. You can do this through OpenEdge Management or OpenEdge Explorer. Or, you can add the following line to the [WebSpeed.Messengers.messenger type] section of the ubroker.properties file:

```
AllowMsngrCmds=1
```

After you have enabled the Administration Utility, use a Web browser to go to the WebSpeed Messenger Administration Page on any machine that has access to the utility. Use a URL with the following format:

```
http://host_name[:port]/scripts_dir/messenger/WService=broker?WSMAdmin
```
When you have accessed the WebSpeed Messenger Administration Page, select **Customize Messenger Error Messages**, which displays the WebSpeed Error Customization Utility shown in the following figure.

**Figure 7: WebSpeed Error Message Customization Utility**

![WebSpeed Error Message Customization Utility](image)

Within the Customization Utility, you can either accept defaults, create your own error messages, or redirect client browsers to a specified URL. Any changes you make are written to the Messenger's *wsCusErr.txt* file, which is created in the Messenger’s log file directory. You do not need to restart the Messenger to make your changes effective.

**Note:** There is a limitation on how many error messages you can customize. Currently, the *wsCusErr.txt* file can contain customizations for no more than 32 error messages.

Another way to customize error messages is to directly edit the *wsCusErr.txt* file, which, if it does not already exist, you can create in the Messenger's log file directory. Use any editor that allows you to save as an unformatted text file.
Entries in the wsCusErr.txt file have the following syntax:

```
error_number 0|1|2
message|URL
```

**error_number**

A number that represents a OpenEdge error. If `error_number` is 0, it will match all OpenEdge error numbers, except those that are specified in the file. You can use an entry with an `error_number` of 0 to create a general message for any errors that you did not explicitly customize.

0|1|2

0 causes the default OpenEdge error message to be displayed. 1 causes the text string specified in the next line to be displayed. 2 causes the URL specified in the next line to be passed to the client browser.

**message|URL**

`message` is a text string representing the customized error message that you want to send to the browser. `URL` is a text string representing the URL for a valid Web site.

An example of the contents of a wsCusErr.txt file follows:

```
8239 1
Name Server is not responding. Please contact administrator.

0 2
http://www.progress.com
```

The first entry causes the specified messages to be displayed whenever an 8239 error is generated. The second entry directs the client browser to the specified Web site whenever any other error is generated.

---

**WebTools**

WebTools are a collection of browser-based utilities that allow you to perform development tasks and to access information. Each tool is directly accessible by clicking a link from the WebSpeed WebTools menu.

You can access WebTools by specifying the WorkShop URL in your browser (`host_name/scripts/cgiip.exe/WService=wsbroker1/workshop`, for example). If you have WebSpeed installed locally, you can also access WebTools from the Tools menu in AppBuilder. Select **Tools > WebTools** from the AppBuilder main window.

The following figure shows the WebTools menu.
The following table gives a brief description of each one of the WebTools, listed in the order that they appear in the WebTools menu. For more detailed information, see the online help, which is accessible from the WebTools menu and from the main frame of each WebTool.

Table 7: WebTools menu items

<table>
<thead>
<tr>
<th>WebTool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Manager</td>
<td>Contains the following utilities:</td>
</tr>
<tr>
<td></td>
<td>• Servers — Manages the WebSpeed brokers, database servers, and other servers</td>
</tr>
<tr>
<td></td>
<td>• CodePath — Defines which directories to include in the code search path, and also allows you to compile all or selected source code files</td>
</tr>
<tr>
<td></td>
<td>• R-Code Library — Packages r-code into procedure library (.pl) files</td>
</tr>
<tr>
<td><strong>WebTool</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Data Browser</strong></td>
<td>Allows you to work with the contents of databases connected to the WebSpeed broker. It contains the following utilities:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Tables</strong> — Lists all the tables in a database, browses through records, and generates reports</td>
</tr>
<tr>
<td></td>
<td>• <strong>Sys</strong> — Shows hidden metaschema database tables</td>
</tr>
<tr>
<td></td>
<td>• <strong>Dict</strong> — Shows a dictionary report of the database</td>
</tr>
<tr>
<td></td>
<td>• <strong>VST</strong> — Shows the virtual system table data generated by the PROMON utility</td>
</tr>
<tr>
<td><strong>Editor</strong></td>
<td>Functions as a basic, single-document text editor. Only one file can be edited at a time. If you attempt to open a new file, the text-entry frame is checked for changes. If there are any changes, you are prompted to save or ignore those changes before a new file is opened.</td>
</tr>
<tr>
<td><strong>File Tools</strong></td>
<td>Provides a directory listing from which you can view, open, run, compile, extract, or delete files in the selected directory. By default, File Tools initially displays a view of your working directory. Once you have created some of your own Web objects and procedure files, you can use the File Tools options to manage and maintain files either in your working directory or in another directory.</td>
</tr>
<tr>
<td><strong>OS Command</strong></td>
<td>Allows you to enter operating system commands. Commands execute in the operating system context of the machine where a WebSpeed Transaction Server is running in development mode. Command output is displayed below the command entry field. The <strong>OS Command</strong> WebTool also allows you to run single-line command scripts. You can create these scripts in any text editor, but the file can only have one line (no carriage returns). Save them in <code>install_dir/src/web/scripts</code> with a <code>.os</code> filename extension. After you create <code>.os</code> files in that directory, their names appear in the OS Command WebTool frame. After you select a script and choose <strong>Load Script</strong>, the script's contents appear in the command entry field. Choose <strong>Submit</strong> to execute. <strong>Note:</strong> WebSpeed does not maintain context from one command to the next. Therefore, some commands (like changing your working directory, for example) do not execute. You get an error message stating that the command generated no output.</td>
</tr>
<tr>
<td><strong>WebTool</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Scripting Lab</strong></td>
<td>Allows you to enter SpeedScript, SQL, JavaScript, VBScript, or HTML code and run it on a WebSpeed Transaction Server that is running in development mode. You can use the <strong>Scripting Lab</strong> to test your code. You can copy your tested code into a Web object (by performing a cut-and-paste operation), but you cannot save code directly in the <strong>Scripting Lab</strong>. The <strong>Scripting Lab</strong> also contains small code samples.</td>
</tr>
</tbody>
</table>
| **Agent Variables** | Displays current environmental variables including:  
  • CGI Environment variables  
  • WebSpeed variables  
  • WebSpeed `WEB-CONTEXT` attributes  
  • WebSpeed `SESSION` attributes  |
| **Databases** | Allows you to look up information about the database schema of any database connected to the WebSpeed agent. The **Databases** tool presents a view of the database on the server machine (which might be a different database from the one the AppBuilder is connected to). You can use the **Databases** tool to view:  
  • Databases connected to the WebSpeed agent  
  • Table definitions for any database connected to the WebSpeed agent  
  • Field and index definitions for a table  
  • Field attributes for a field  
  • Index attributes for an index  |
| **Messages** | Allows you to access information about error messages generated by WebSpeed. When you enter a message number, it displays a description of the message.  |
| **Object State** | Allows you to view the state of procedures that a WebSpeed agent is running.  
  When you select an object state link, a table appears containing the name of each `persistent object` running on the WebSpeed agent and a description of the type and state of the object.  |
| **ProPath** | Displays a list of the directories contained in the `PROPATH` environment variable. The `PROPATH` environment variable specifies where the WebSpeed agent will search to locate the file it has been requested to run. You can use the **ProPath** tool to view and verify the directories included in the `PROPATH`. You can also use the **ProPath** tool to open a directory in the **File Tools** tool. |
### Language support

The following sections discuss parts of the ABL and OpenEdge environment that are either specific to WebSpeed or useful in designing WebSpeed applications.

### SpeedScript and CGI wrappers

SpeedScript is an implementation of the ABL that is primarily used to develop Web applications. ABL is inserted into HTML by using special HTML tags. When the WebSpeed compiler compiles this HTML, it converts it into a CGI wrapper automatically. You can also write CGI wrappers directly. SpeedScript includes a built-in Web server input/output (I/O) system that provides access to the CGI environment and Web page output stream for a Web request.

**Note:** SpeedScript supports extensions that allow the use of XML through SAX and the Document Object Model (DOM) interface. These extensions provide the basic input, output, and low-level data manipulation capabilities required to use data contained in XML documents. For more information about XML support, see *OpenEdge Development: Working with XML*, which describes XML support in the context of the ABL. However, the information also applies to SpeedScript, which is based on the ABL.

---

### WebTool

<table>
<thead>
<tr>
<th>WebTool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual System Tables</td>
<td>Allows you to examine the status of a database and monitor its performance.</td>
</tr>
<tr>
<td>Developer’s Corner</td>
<td>Links to the Progress Communities WebSpeed Forum: <a href="http://communities.progress.com/pcom/community/psdn/openedge/webspeed">http://communities.progress.com/pcom/community/psdn/openedge/webspeed</a></td>
</tr>
<tr>
<td>Help</td>
<td>Contains links to WebTools help topics.</td>
</tr>
</tbody>
</table>
See the following sources for more information regarding SpeedScript:

- *OpenEdge Development: ABL Reference* — Describes the syntax of all SpeedScript language elements; also identifies which language elements apply to the ABL (Advanced Business Language), to SpeedScript, or to both the ABL and SpeedScript
- *OpenEdge Getting Started: ABL Essentials* — Covers important programming concepts such as database locking and transaction rules, program block and resource scoping rules, and the use of persistent procedure

**Similarities between the ABL and SpeedScript**

The similarities between the ABL and SpeedScript include:

- Block structure and resource scoping rules are the same
- Database events (such as `CREATE` and `DELETE`) can be handled in the same way
- Database locking and database transaction rules are the same
- Both can use SmartDataObjects as data sources
- Both can use the AppServer to process requests
- Both can be written using the same AppBuilder tools (Procedure Window, the Section Editor, and the TreeView)

**Differences between the ABL and SpeedScript**

Some important differences between SpeedScript programming and ABL programming are:

- ABL applications are usually state-aware, while SpeedScript applications are often stateless. The distinction between state-aware and stateless applications is briefly discussed in *OpenEdge Application Server: Developing WebSpeed Applications*.
- GUI widget events are not used in WebSpeed SpeedScript applications. Visual elements are handled by HTML, rather than as GUI widgets.
- The preprocessor `{&OUT}` statement is used to output data to the HTML page, rather than the `DISPLAY` statement. For information, see *OpenEdge Application Server: Developing WebSpeed Applications*.
- In SpeedScript, all terminal-focused I/O is replaced by a block mode Web-oriented I/O, where the SpeedScript frame is the block data structure. Therefore, I/O blocking statements, such as `UPDATE` or `PROMPT-FOR`, cannot be used in SpeedScript applications. Similarly, the I/O blocking options of statements like the `MESSAGE (VIEW-AS ALERT-BOX)` and `PAUSE (MESSAGE)` statements are ignored.
- Very few ABL events apply to WebSpeed applications, except database events. The one essential event in SpeedScript is `WEB-NOTIFY`. However, in normal use, this event is handled exclusively by the agent control program (`web-disp.p`).
- `WAIT-FOR` cannot be used for user input in SpeedScript applications.
- Some procedures that are available through include files and the Insert Call button of the Section Editor are only appropriate for WebSpeed applications. Some of these are:
  - `hidden-field-list`
• set-cookie
• get-cookie

See the WebSpeed API reference in the online help for a list of the public APIs.

• SpeedScript includes special extensions, including a virtual Web output device ("WEB") to
define Web page output streams to your Web server and the WEB-CONTEXT system handle to
access the request environment. However, most of these extensions are wrapped in the API
functions, method and event procedures, and preprocessor definitions provided with WebSpeed.

These examples also rely on SpeedScript preprocessor references, especially \{&OUT\} and
{&DISPLAY}, to direct output to the WebSpeed-defined output stream, WebStream. You can find
the definitions for these preprocessor references (and several others) in
install-path/src/web/method/cgidefs.i. For more information on the {&DISPLAY}
preprocessor reference, see OpenEdge Application Server: Developing WebSpeed Applications.

SpeedScript versus JavaScript

It is a common practice to use both SpeedScript and JavaScript when developing WebSpeed
applications. SpeedScript has advantages for developing the business logic of applications, while
JavaScript is a good programming tool for adding user interface elements to Web applications.

If you use either the Report or Detail templates in AppBuilder to create a WebSpeed Web object,
you can view the resulting HTML source file and see a combination of SpeedScript and JavaScript.
The templates will help you create SpeedScript to implement database queries and updates, and
they will create JavaScript event handlers (like onMouseOver, onClick, etc.) to implement
interactive features of the WebSpeed applications.

The <SCRIPT> tag for JavaScript employs the same syntax as the <SCRIPT> tag for Embedded
SpeedScript, as shown:

```html
<SCRIPT Language="JavaScript">
JavaScript Code
</SCRIPT>
```

In some situations, you do not need a <SCRIPT> tag. JavaScript event handlers, for example, do
not require a <SCRIPT> tag when they are used as an attribute to an HTML tag, as shown in the
following example:

```html
<BODY onLoad="alert('Done');">
```

Some other factors that you should keep in mind when using JavaScript in WebSpeed applications
are:

• End users of your WebSpeed application will be able to see your JavaScript code when they
view HTML source in their browsers. They can see the HTML output that Embedded SpeedScript
generates, but they do not see the actual SpeedScript source code. (This is because the
SpeedScript code executes on the server side while the JavaScript executes on the client-side
browser.)

• No static or dynamic HTML can be generated from the JavaScript code that is between HTML
<SCRIPT> tags.
SpeedScript is executed on the server side by the WebSpeed agent. JavaScript is executed on the client side by the Web browser.

### Additional SpeedScript components

The WebSpeed development environment also contains the following components:

- **Method procedures** — A set of standard SpeedScript procedures that provides a variety of services to Web objects. Some method procedures that duplicate API functions are supported for upward compatibility with earlier WebSpeed versions. Others provide unique services to some types of Web objects, such as main-lining the response to Web requests. Some of these you can customize for each Web object. The source for many of the standard method procedures resides in the utility object `web-util.p` under `install-ath/src/web/objects`. The `run-web-object` procedure follows a protocol designed to ensure the integrity of the Web object, whether it is stateless or state-aware. This allows WebSpeed to manage Web objects in a consistent manner. The `run-web-object` method procedure is defined in the utility object, `install-path/src/web/objects/web-util.p`.

- **Control handlers** — A special class of procedures that execute in response to pseudo events in HTML-mapping Web objects, and that you can customize. These pseudo event procedures include `web.input` and `web.output`. By overriding `web.input` and `web.output`, you can replace the default data movement between HTML field definitions and corresponding field objects for HTML-mapping Web objects. The default versions of these control handlers for each type of HTML field are located in the `tagmap` utilities under `install-path/src/web/support`. Each utility procedure understands how to convert data between a particular type of HTML form element and a SpeedScript field object of the corresponding type.

- **tagmap.dat** — A file that contains default mappings between HTML form element types and SpeedScript field object types for HTML-mapping Web objects. Each entry in the file includes the location of the default `web.input` and `web.output` control handler procedure for the field mapping. This file is also where you can define your own mappings and custom tags for your application. This file resides in your WebSpeed installation directory (`install-path/`).

- **web-disp.p** — The control program that runs on all WebSpeed agents and executes all Web objects. The SpeedScript source resides in `install-path/src/web/object`. It manages various transaction states that can affect the whole application. This SpeedScript procedure is included with the development environment because it is central to the operation of WebSpeed applications. How `web-disp.p` manages Web objects can affect how and when you might set and evaluate transaction states in each Web object. This program also initializes the utility object `web-util.p` (residing in the same directory), where most API functions and method procedures reside at run time. WebSpeed has a special set of procedure-calling conventions. The first convention relies on the `run-web-object` method procedure. This procedure is the standard method to execute a Web object from within another procedure. It is also the basic method `web-disp.p` uses to execute Web objects in response to Web requests.

### WebSpeed global variables

The WebSpeed global variables are a rich set of variables available to all Web objects running on the same agent that maintains the current CGI environment and other information about the current Web request or state-persistent WebSpeed transaction. The definitions of these variables reside in `install-path/src/web/method/cgidefs.i`. 

---

**OpenEdge Getting Started: WebSpeed Essentials**
WebSpeed preprocessors

The preprocessor is a function of the ABL compiler that also applies to SpeedScript. On its initial pass through source code, the compiler looks for preprocessor directives and performs text substitutions when it finds them. All directives begin with an ampersand (&).

The WebSpeed preprocessors, which are listed in the following table, provide consistent access to the Web environment, especially the Web output stream. The definitions of WebSpeed preprocessor names reside in `install-path/src/web/method/cgidefs.i`.

Table 8: WebSpeed preprocessors

<table>
<thead>
<tr>
<th>Preprocessor name</th>
<th>Assigned value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;WEBSTREAM</td>
<td>STREAM Webstream</td>
</tr>
<tr>
<td>&amp;OUT</td>
<td>PUT {&amp;WEBSTREAM} UNFORMATTED</td>
</tr>
<tr>
<td>&amp;OUT-FMT</td>
<td>PUT {&amp;WEBSTREAM}</td>
</tr>
<tr>
<td>&amp;OUT-LONG</td>
<td>EXPORT {&amp;WEBSTREAM}</td>
</tr>
<tr>
<td>&amp;DISPLAY</td>
<td>DISPLAY {&amp;WEBSTREAM}</td>
</tr>
</tbody>
</table>

WebSpeed API

WebSpeed APIs include a set of standard WebSpeed functions (user-defined SpeedScript functions) that provides a variety of services to Web objects. API functions handle low-level Web object tasks such as formatting URLs and returning specific values from the CGI environment. All API functions are available for your application. The source resides in several include (.i) files under `install-path/src/web/method`, including `cgiutils.i`, `cookies.i`, and `message.i`. Access to these functions in a Web object is provided by including `install-path/src/web/method/cgidefs.i`.

For more information on WebSpeed API functions, see the WebSpeed API appendix in *OpenEdge Application Server: Developing WebSpeed Applications*. This information is also available through the AppBuilder online help. Choose Help > Help Topics from the AppBuilder menu bar. Then select the Find tab and type WebSpeed API in the top field of the dialog box.
XML

Extensible Markup Language (XML) is supported directly in WebSpeed's SpeedScript (ABL) language, allowing you to exchange data between OpenEdge-supported data sources and XML documents. XML, considered the standard for exchanging data between disparate applications, is a markup language like HTML. However, unlike HTML, XML describes document content in terms of the data without regard for how it is to be displayed.

WebSpeed contains an industry standard XML parser, allowing developers to use SpeedScript to create programs that send and receive XML documents to/from other XML-enabled Web applications using Document Object Model (DOM) and SAX interfaces.

JSON

JavaScript Object Notation (JSON) is supported directly in WebSpeed's SpeedScript (ABL) language, allowing you to exchange data between OpenEdge-supported data sources and JSON data. Specifically, you can serialize ABL ProDataSets and temp-tables to and from JSON data.

JSON developers use JSON as an alternative data interchange format to XML. XML is widely used to exchange data in a heterogeneous environment. However, some developers consider XML as too verbose for exchanges between a web browser and a web server as part of a rich internet application. For example, JSON is widely used in AJAX applications.

WebSpeed source files

All SpeedScript source files provided with WebSpeed, including web-disp.p, web-util.p, example Web objects, and other SpeedScript source files, reside in directories under install-path/src/web and install-path/src/web2.

The source files in install-path/src/web are described as follows:

- **Examples** — Contains the source for Web object examples described in this manual, including HTML and offset files. It also contains additional examples of interest.

- **Method** — Contains the source for various SpeedScript compile-time include files (similar in function to C include files) that define some of the method procedures and API functions used by WebSpeed to construct Web objects.

- **Objects** — Contains the source for the main Web object dispatch procedure web-disp.p and the utility object web-util.p.

- **Support** — Contains all of the tagmap utility procedures for HTML-mapping Web objects that are specified in tagmap.dat and that define the default web.input and web.output control handlers for supported HTML form element and SpeedScript field object types. It also contains some run-time procedures for debugging and interpreting the offset file for an HTML-mapping Web object.

- **Template** — Contains skeleton files used to create new Web objects and other types of support objects and files, including:
  - A file that defines the New File links that use these templates in the AppBuilder Files component (web.cst).
  - A blank embedded SpeedScript file template (script.html)
  - An embedded SpeedScript template for creating a database browser (browse.html)
An embedded SpeedScript template for creating an application welcome page (main.html)

A HTML frame set template (frameset.html).

An embedded SpeedScript template for generating a table of database fields that can be executed by another Web object (table.html)

A SpeedScript procedure file template (procedur.p)

A SpeedScript include file template (include.i)

A CGI Wrapper template for creating a SpeedScript-generating Web object (wrap-cgi.w).

A HTML-mapping template and include files (html-map.w and hmapmain.i) for creating an HTML-mapping Web object

The source files in `install-path/src/web2` are described as follows:

- **Super procedures** — These are .i and .p files that contain the definitions of WebSpeed super procedures. To use super procedures, you include the .i file in your ABL code rather than running the .p file directly. Super procedures are in the `install-path/src/web2` directory.

  For more information about super procedures, see *OpenEdge Deployment: Managing ABL Applications*.

- **Templates** — Code relating to WebSpeed Report, Detail, and HTML Mapping wizards is in the `install-path/src/web2/templates` directory. These files can be used as models for creating custom templates that can be made accessible to the AppBuilder through a .cst file.

  There are two kinds of files in the template directory. First are the .w and .i template files. These are used to create new objects that can be made available from the AppBuilder palette. More information on using .w and .i template files to extend AppBuilder can be found in *OpenEdge Application Server: Developing AppServer Applications*.

  The second type of file is the HTML template files (.dat files) for the report and detail wizards. These can be used as examples of how to create new wizards that create embedded ABL files (.html files). Each .dat file has a companion .w file with the same filename (for example, wreport.w and wreport.dat) that acts as a SpeedScript wrapper for the .dat file.
Running and Deploying WebSpeed Applications

This chapter includes information about deploying WebSpeed, including important security considerations, as described in the following sections:

For details, see the following topics:

• Distributed WebSpeed environments
• WebSpeed security
• Firewall configuration and debugging
• Optimizing WebSpeed performance
• Running sample applications

Distributed WebSpeed environments

The pieces of the WebSpeed architecture do not have to reside on a single machine. However, there are dependencies that you must consider before installing your own WebSpeed environment. The way you choose to distribute your environment might affect how your application performs. You should consider this during the early stages of your design process, as well as the later stages when you are planning your deployment strategy.
The following sections show sample WebSpeed environments and briefly describe how they should be installed and managed. They concentrate on the WebSpeed components, but there are other considerations as well. For example, Securing your WebSpeed server machine on page 93 discusses the addition of firewalls to a WebSpeed environment.

After you have installed your environment, you must consider the interdependencies when you write startup routines for your environment. Some components cannot start correctly unless they can communicate with some other component. These dependencies shape proper startup routines. The brief startup sequences in the following sections can help you avoid common errors that occur when components are started before everything they depend on has started.

**Note:** Some components can be configured to start automatically when other components start.

For example, before launching a WebSpeed application, you should make a data source available to the WebSpeed broker and agents for your application. When you shut down your application, the broker and agents should shut down before the databases. If agents lose their database connections prematurely, you might have to complete the shutdown manually.

In most cases, when a machine or component fails, you only have to restart that machine or component. If your data source in a complex distributed configuration fails, data integrity concerns might make it necessary to bring down other components before restarting your data source.

The examples of WebSpeed configurations in this section are meant to illustrate which components must reside on the same machine and how to establish links between the various WebSpeed components on different machines. You can adapt these examples to your needs, as described in the following sections.

**Single-machine configuration**

The most basic configuration for WebSpeed is to install the WebSpeed Workshop application on the same machine as your Web server and a data source. The WebSpeed Workshop uses a subset of the OpenEdge development tools focused on developing and testing WebSpeed applications. With the Workshop, you do not have access to the graphical interface design capabilities of the AppBuilder.

The following figure shows the components installed in a single-machine configuration.

**Note:** The WebSpeed Broker and the WebSpeed Agents it controls are collectively referred to as the WebSpeed Transaction Server in this figure.
The WebSpeed Workshop includes the WebSpeed Development Server, which supports a single developer. The WebSpeed Development Server comes with only two WebSpeed Agents. The WebSpeed Development Server includes the Progress OpenEdge Personal RDBMS. The OpenEdge Personal RDBMS can handle up to five local connections: one for the WebSpeed AppBuilder, one for OpenEdge Management or OpenEdge Explorer, and three for the applications that you are developing. This is suitable for the needs of a single developer.

As the previous figure shows, the Web server, the WebSpeed WebTools, and the WebSpeed Messenger form an interdependent unit. While the Web server can handle non-WebSpeed traffic in addition to the WebSpeed traffic, the WebSpeed WebTools and Messenger cannot operate independently from the Web server.

**Setting up a single-machine configuration**

To set up a single-machine configuration:

1. Install a Web browser and a Web server on your machine.
2. Perform a Complete Installation of WebSpeed Workshop onto your machine.

**Startup sequence**

After you have installed and configured everything, you can start your WebSpeed environment. To start a single machine configuration:

1. Start the AdminServer.

   **Note:** Once the AdminServer has started, you can use OpenEdge Management or OpenEdge Explorer.

2. Start the data source in multi-user mode.
3. Start the NameServer.

   **Note:** The default setting for the NameServer is to start when the AdminServer starts.
4. Start the WebSpeed Transaction Server.
5. Start the Web server.
6. Start the Web browser by choosing the WebTools from the AppBuilder's Tools menu as needed.

Development network with a central WebSpeed machine

A common WebSpeed configuration is a network where a central machine handles most of the WebSpeed components. Individual developers only need the WebSpeed AppBuilder and a Web browser on their workstations.

The following figure shows the components installed in a development network with a central machine.

**Note:** The WebSpeed Broker and the WebSpeed Agents it controls are collectively referred to as the WebSpeed Transaction Server in this figure.

**Figure 10: Development network with a central WebSpeed machine**

This configuration uses the WebSpeed Transaction Server and some data source on the central machine and the WebSpeed Workshop on the workstations. This figure shows a central machine, Mars, configured to support two development workstations, Phobos and Deimos.

The WebSpeed Transaction Server supports a team of developers creating WebSpeed applications. You can start up as many WebSpeed Agents as you have licenses for. Depending on the locking strategies for WebSpeed Agents you use in your applications, any given Agent might serve the needs of several developers or only one developer. Because the WebSpeed Transaction Server does not include a database, you must install either the Progress OpenEdge Workgroup RDBMS or the appropriate DataServer to connect to your non-OpenEdge database.

Setting up a central development machine configuration

To set up a central development machine configuration:
1. Install a Web server on Mars.
2. Install the WebSpeed Transaction Server on Mars.
3. Install the appropriate components for your intended data source on Mars:
   • If you use the OpenEdge Database, install the OpenEdge Workgroup RDBMS.
   • If you connect to a non-OpenEdge data source, install the appropriate OpenEdge DataServers for your data source. See the appropriate OpenEdge DataServer guide for details.
4. Install the WebSpeed Workshop, which includes the AppBuilder, on Phobos and Deimos.

**Beyond the basic installation**

If you are going to use a non-OpenEdge database, you must install the appropriate DataServer on the Mars server.

**Startup sequence**

After you have installed and configured everything, you can start your WebSpeed environment.

To start a central development machine configuration:

1. Start the AdminServer on Mars.
2. Start the data source (and the data server broker, if needed) on Mars in multi-user mode.
3. Start the NameServer on Mars.
4. Start the WebSpeed Transaction Server on Mars.
5. Start the Web server on Mars.
6. Start a WebSpeed AppBuilder on a workstation (Phobos and Deimos) as needed.
7. Start a Web browser by choosing the WebTools from the AppBuilder's Tools menu on a workstation as needed.

**Development network with a dedicated Web server**

Another common configuration is a network made up of workstations; a machine that handles the Web server, WebSpeed WebTools, and WebSpeed Messenger; and a machine that handles the WebSpeed Transaction Server, the applications, and data source.

The following figure shows a sample configuration for a WebSpeed development network with a dedicated Web server.

**Note:** The WebSpeed Broker and the WebSpeed Agents it controls are collectively referred to as the WebSpeed Transaction Server in this figure.
This configuration has the WebSpeed Messenger installed on the Web server's machine, Europa. The host machine, Jupiter, uses the WebSpeed Transaction Server. Because the WebSpeed Transaction Server does not include a database, you must install the OpenEdge Workgroup RDBMS or the appropriate DataServer to connect to your non-OpenEdge database.

One of the workstations, Ganymede, has OpenEdge Studio installed to gain access to the full capabilities of the AppBuilder. The developer using Ganymede can alternate between developing Web applications and non-Web-based applications. The other workstation, Callisto, has the WebSpeed Workshop installed.

**Setting up a development network with a dedicated Web server configuration**

To set up a development network with a dedicated Web server configuration:

1. Install a Web server on Europa.
2. Install the WebSpeed Transaction Server on Jupiter.
3. Install the appropriate components for your intended data source on Jupiter:
   - If you use the OpenEdge Database, install the OpenEdge Workgroup RDBMS.
   - If you connect to a non-OpenEdge data source, install the appropriate OpenEdge DataServers for your data source. See the appropriate OpenEdge DataServer guide for details.
4. Install the WebSpeed Messenger on Europa.

**Note:** The WebSpeed Messengers for all platforms are available from the Download Center at [http://www.progress.com](http://www.progress.com).

5. Install OpenEdge Studio on the Ganymede workstation.
6. Install the WebSpeed WorkShop on the Callisto workstation.
Beyond the basic installation

If you are going to use a non-OpenEdge database, you must install the appropriate DataServer on the Jupiter server.

Startup sequence

After you have installed and configured everything, you can start your WebSpeed environment. To start a development network with a dedicated Web server configuration:

1. Start the AdminServer on Jupiter.

   Note: Once the AdminServer has started, you can use OpenEdge Management or OpenEdge Explorer to start other components on Jupiter.

2. Start the data source (and the data server broker, if needed) on Jupiter in multi-user mode.
3. Start the NameServer on Jupiter.
5. Start the Web server on Europa.
6. Start an AppBuilder on a workstation (Ganymede and Callisto) as needed.
7. Start a Web browser on a workstation by choosing the WebTools from the AppBuilder's Tools menu as needed.

Multiple LAN development environment

Some development environments include several local area networks (LANs). For example, a company's intranet might include two LANs—a Sales LAN and a Manufacturing LAN.

The common OpenEdge administration architecture includes a method for connecting NameServers to other NameServers. When a NameServer receives a request for an application service, it checks to see whether any of the WebSpeed brokers registered with it supports that application service. If none of its registered brokers supports that application service, the NameServer passes the request to other NameServers that it knows about.

Using this capability, you can link OpenEdge installations across the LANs that make up a wide area network. The following figure shows the products installed on a multiple LAN configuration.
In this configuration, the NameServer machine, Saturn, is the central communication point that routes requests to the WebSpeed Transaction Servers on both LANs. Requests start from workstations, like Prometheus (which has OpenEdge Studio installed on it). A workstation passes the request to the Web server machine, Pandora, which has the WebSpeed Messenger installed. The Messenger asks the NameServer on Saturn to find a WebSpeed Transaction Server that can handle the request, by specifying an application service with the \texttt{WService} parameter.

Calypso and Hyperion both have the Transaction Server installed. The Transaction Servers are registered with Saturn's NameServer. Saturn's NameServer checks its list of registered Transaction Servers to see whether one supports the application service that can handle the request from Pandora's Messenger. If one of them does, the NameServer passes its location back to the Messenger and the request is completed. The Transaction Server installed on Titan includes a NameServer. You can register this NameServer with Saturn's NameServer as a NameServer Neighbor. If the WebSpeed Brokers on Calypso or Hyperion do not support the correct application service, Saturn's NameServer passes the request to the NameServer on Titan to see whether the Broker registered with it supports that application service. If the Broker on Titan does support the application service, Titan's NameServer passes its location to Pandora's Messenger. Then the request is completed as normal.
For more information on how this works, see the section on using NameServer neighbors in *OpenEdge Getting Started: Installation and Configuration*.

Note that there are only two machines with data sources in this configuration—Hyperion, on the Windows side, and Atlas, on the UNIX side. Because the WebSpeed Transaction Server does not come with a database, you must install the Progress OpenEdge Enterprise RDBMS or the appropriate DataServer to connect to your non-OpenEdge database. The WebSpeed Agents on any of the Transaction Server machines in the configuration can access either data source.

## Setting up a multiple LAN configuration

To set up a multiple LAN configuration:

1. Install a Web server on Pandora.
2. Install the NameServer on Saturn.
3. Install the WebSpeed Transaction Server on Calypso, Hyperion, and Titan.
4. Install the WebSpeed Messenger on Pandora.
5. Install OpenEdge Studio or the WebSpeed Workshop on Prometheus and your other workstations, according to your development needs.
6. Install the appropriate data source components on Hyperion and Atlas, depending on what you use as your data source:
   - If you use the OpenEdge Database, install the OpenEdge Enterprise RDBMS.
   - If you connect to a non-OpenEdge data source, install the appropriate OpenEdge DataServers. See the appropriate OpenEdge DataServer guide for details.

## Beyond the basic installation

When you expect heavy loads on your network or want to increase its fault tolerance, you might want to add the load balancing option for the NameServer. If you deploy several WebSpeed Transaction Servers that support the same application service, you can use load balancing to control how the load is spread over them. See *Understanding the NameServer's load balancing option* on page 39.

If you are going to use a non-OpenEdge database, you must install the appropriate DataServer on the Atlas and Hyperion servers.

## Startup sequence

After you have installed and configured everything, you can start your WebSpeed environment.

To start a multiple LAN configuration:

1. Start the AdminServers on Saturn, Calypso, Hyperion, Titan, and Atlas.

   **Note:** Once the AdminServers have started, you can use OpenEdge Management or OpenEdge Explorer from any machine to start other components on these machines.

2. Start the data sources (and data server brokers, if needed) on Hyperion and Atlas in multi-user mode.
3. Start the NameServers on Saturn and Titan.
5. Start the Web server on Pandora.
6. Start an AppBuilder on a workstation (like Prometheus) as needed.
7. Start a Web browser on a workstation by choosing the WebTools from the AppBuilder's Tools menu as needed.

**Note:** If one of the data sources fails, you should stop all the WebSpeed Transaction Servers that support applications that use that data source. Bring the data source back up and then restart the Transaction Servers. This helps ensure data integrity.

**Deployment configuration with a dedicated Web server**

When you deploy your applications, you do not need all the components that usually exist in a development configuration.

The following figure shows the products installed in a deployment network with a dedicated Web server.

**Note:** The WebSpeed Broker and the WebSpeed Agents it controls are collectively referred to as the WebSpeed Transaction Server in this figure.

**Figure 13: Deployment configuration with a dedicated Web server**

This configuration has the WebSpeed Messenger product installed on the Web server's machine, Oberon. The main machine, Uranus, has the WebSpeed Enterprise Transaction Server and some data source. The end users only need a Web browser to link up to your applications; they do not need any OpenEdge product installed on their machines.
Setting up a deployment network with a dedicated Web server

To set up a deployment network with a dedicated Web server:

1. Install a Web server on Oberon.
2. Install the WebSpeed Messenger on Oberon.

**Note:** Since the WebSpeed WebTools are only meant for developers, you should remove access to them. During the Messenger-only installation, do not choose the option to either create a virtual directory or copy the static HTML files.

3. Install the WebSpeed Enterprise Transaction Server on Uranus.
4. Install the appropriate components for your intended data source on Uranus:
   - If you use the OpenEdge Database, install the OpenEdge Enterprise RDBMS.
   - If you connect to a non-OpenEdge data source, install the appropriate OpenEdge DataServers for your data source. See the appropriate OpenEdge DataServer guide for details.
5. Install your r-code on Uranus and include its location in the WebSpeed agent's PROPATH.
6. Copy any Java applet, static HTML, and .gif files needed for your applications to the Web server's /docroot/webspeed directory on Oberon.

Beyond the basic installation

As described, this configuration uses the components for a heavy load system. If you expect a light load on the network, you might consider installing the WebSpeed Transaction Server and Workgroup RDBMS instead of the Enterprise Transaction Server and Enterprise RDBMS. The WebSpeed Transaction Server can handle up to 50 WebSpeed Agents. The Workgroup RDBMS can handle up to 65 connections.

When you expect heavy loads on your network or want to increase its fault tolerance, you might want to add the load balancing option for the NameServer. If you deploy several WebSpeed Transaction Servers that support the same application service, you can use load balancing to control how the load is spread over them. See Understanding the NameServer's load balancing option on page 39.

If you are going to use a non-OpenEdge database, you must install the appropriate DataServer on the Uranus server.

Startup sequence

After you have installed and configured everything, you can start your WebSpeed environment.

To start a deployment network with a dedicated Web server:

1. Start the AdminServer on Uranus.

   **Note:** Once the AdminServer has started, you can use OpenEdge Management or OpenEdge Explorer to start other components on Uranus.

2. Start the data source (and the data server broker, if needed) on Uranus in multi-user mode.
3. Start the NameServer on Uranus.
4. Start the WebSpeed Transaction Server on Uranus.
5. Start the Web server on Oberon.

Development and deployment shared configuration

When a company develops its own applications, it can have networks that are used for development, testing, and deployment simultaneously.

The following figure shows the products installed in a shared development and deployment network.

**Note:** The WebSpeed Broker and the WebSpeed Agents it controls are collectively referred to as the WebSpeed Transaction Server in this figure.

Figure 14: Development and deployment shared configuration

In this configuration, the dedicated Web server machine, Neptune, has two Web servers installed. One Web server handles the development workload, and the other handles the deployment workload. This allows the developers to experiment with the environment without bringing down the deployment side. Each of the Web servers requires its own WebSpeed Messenger, but only the development Web server needs the WebSpeed WebTools.
On the development side, the Messenger routes requests to the NameServer on Triton. Triton's NameServer only has the WebSpeed Transaction Servers on Proteus registered, so it looks there for a Transaction Server to handle the request. Proteus has the WebSpeed Enterprise Transaction Server and either the OpenEdge Enterprise RDBMS or a non-Progress data source installed on it.

On the deployment side, that Messenger routes requests to the NameServer on Galatea. Galatea has the WebSpeed Enterprise Transaction Server installed. The deployment side uses a dedicated machine, Nereid, to hold its data source (that is, the OpenEdge Enterprise RDBMS or a non-Progress data source).

**Setting up a development and deployment shared configuration**

To set up a development and deployment shared configuration:

1. Install two copies of your Web server on Neptune.
2. Install the Enterprise NameServer on Triton.
3. Install the WebSpeed Enterprise Transaction Server on Proteus.
4. Install the WebSpeed Messenger on Neptune using the development Web server's Document Root directory and scripts path.
5. Configure the Development Web server's WebSpeed Messenger on Neptune to pass requests to Triton's NameServer.
6. Install OpenEdge Studio or the WebSpeed Workshop on Thalassa and your other workstations, according to your development needs.
7. Install the WebSpeed Messenger on Neptune using the deployment Web server's Document Root directory and scripts path.

**Note:** Since the WebSpeed WebTools are only meant for developers, you should remove access to them through the deployment Web server. During the Messenger-only installation, do not choose the option to either create a virtual directory or copy the static HTML files.

8. Install the WebSpeed Enterprise Transaction Server on Galatea.
9. Configure the Deployment Web server's WebSpeed Messenger on Neptune to pass requests to Galatea's NameServer.
10. Install the appropriate data source components on Proteus and Nereid. This depends on what you are planning to use as your data source:
    - If you use the OpenEdge Database, install the OpenEdge Enterprise RDBMS.
    - If you connect to a non-OpenEdge data source, install the appropriate OpenEdge DataServers. See the appropriate OpenEdge DataServer guide for details.
11. Install your r-code on Galatea and include its location in the WebSpeed agent's `PROPATH`.
12. Copy any Java applet, static HTML, and .gif files that your applications must the deployment Web server's `/docroot/webspeed` directory on Neptune.
Beyond the basic installation

As described, this configuration uses the components for a heavy load system. If you expect a light load on the network, you might consider installing the WebSpeed Transaction Server and OpenEdge Workgroup RDBMS instead of the Enterprise Transaction Server and OpenEdge Enterprise RDBMS. The WebSpeed Transaction Server can handle up to 50 WebSpeed Agents. The OpenEdge Workgroup RDBMS can handle up to 65 connections.

You can expand the capacity of either side of this configuration by adding new server machines with the same products installed. Register the WebSpeed Transaction Servers on the new machines with the NameServer that is handling the Transaction Servers for that side.

When you expect heavy loads on your network or want to increase its fault tolerance, you might want to add the load balancing option for the NameServer. If you deploy several WebSpeed Transaction Servers that support the same application service, you can use load balancing to control how the load is spread over them. See Understanding the NameServer’s load balancing option on page 39 for additional information.

If you are going to use a non-OpenEdge database, you must install the appropriate DataServer on the Nereid and Proteus servers.

Development startup sequence

After you have installed and configured everything, you can start your WebSpeed environment.

To start the Development side of this shared configuration:

1. Start the AdminServers on Triton and Proteus.

   **Note:** Once the AdminServers have started, you can use OpenEdge Management or OpenEdge Explorer from any machine to start other components on these machines.

2. Start the data source (and the data server broker, if needed) on Proteus in multi-user mode.
3. Start the NameServer on Triton.
5. Start the development Web server on Neptune.
6. Start an AppBuilder on a workstation (like Thalassa) as needed.
7. Start a Web browser on a workstation by choosing the **WebTools** from the AppBuilder’s **Tools** menu as needed.

Deployment startup sequence

After you have installed and configured everything, you can start your WebSpeed environment.

To start the Deployment side of this shared configuration:

1. Start the AdminServers on Galatea and Nereid.

   **Note:** Once the AdminServers have started, you can use OpenEdge Management or OpenEdge Explorer from any machine to start other components on these machines.

2. Start the data source in multi-user mode on Nereid.
3. Start the NameServer on Galatea.
4. Start the WebSpeed Transaction Server on Galatea.
5. Start the deployment Web server on Neptune.

Note: If one of the data sources fails, you should stop all the WebSpeed Transaction Servers supporting applications that use that data source. Bring the data source back up and then restart the Transaction Servers. This helps ensure data integrity.

WebSpeed security

WebSpeed is an application service technology for Web browser applications. OpenEdge supports a number of special-purpose security options for WebSpeed that are tailored for the WebSpeed environment, including:

- Options for different working models, including Evaluation, Development, and Production that provide different levels of access to the WebSpeed Web Tools.

- Configuration options to minimize security risks, including options for:
  - Special port numbers and WebSpeed server names
  - Minimizing the number of PROPATH entries
  - Minimizing unauthorized access to WebSpeed Messengers
  - Restricting file upload directories
  - Additional WebSpeed secure configuration considerations

- Password authentication using SpeedScript

- Data privacy over the connection between the WebSpeed Messenger and the WebSpeed Transaction Server using SSL

Note: WebSpeed agents, as ABL (Advanced Business Language) database clients, can also access the OpenEdge RDBMS using SSL.

- Maximizing compatibility with firewalls, including:
  - TCP/IP port management
  - Network Address Translation (NAT) and distributed configurations
  - NameServer client port range value settings
  - Direct connection to the WebSpeed Transaction Server without a NameServer to avoid UDP conflicts with firewalls
Making your application secure

A Web user can potentially run any procedure file that is accessible from an agent’s PROPATH. This includes any procedures in your application working directory and any procedure files that are relative to the install-path directory. The Web user should be restricted from gaining access to this functionality. For example, the procedure can compile and run WebSpeed programs typed in by a Web user. If you make this procedure available to a Web user, you essentially give that Web user complete control over an agent. A number of SpeedScript statements would even give the Web user access to your operating system's file structure.

There are a number of things you can do to avoid this possibility. One approach is to start your agents with the run-time client (-rr) startup parameter. This parameter ensures that agents can only run precompiled code. This allows you to leave uncompiled procedures on the PROPATH without concern that they can be run from a Web browser. However, this approach does not allow you to take advantage of WebSpeed's compile-time flexibility. Depending on how you want to write your application, this might be important.

You can also use the check-agent-mode API function to allow some routines to work for Development but not for Production. For more information on the environment options, see the UNIX ubroker.properties.README file, or see OpenEdge Application Server: Developing WebSpeed Applications.

Yet another approach is to move any procedures off of the PROPATH that you do not want a Web user to run. For example, if you do not want a Web user to run the runscrpt.w procedure, then you must move it into a directory that is not included on the PROPATH and is not relative to the PROPATH.

Caution: One of the most important security considerations is to deny end users access to WebTools. Access to WebTools allows users to run utilities that can potentially alter or damage your system. Therefore, you should make sure that your WebSpeed agents run in production mode for deployed applications. None of the WebTools can run in production mode.

By default, the WebSpeed agents run in development mode. You can change to production mode by using OpenEdge Management or OpenEdge Explorer to change the agent’s properties. See OpenEdge Application Server: Developing WebSpeed Applications for more information.

The following aspects of your configuration should be secure when deploying WebSpeed:

• Network traffic
• Web server
• WebSpeed server
• Application

The following sections briefly discuss each of these topics. Security is a broad and complex topic. You might want to consult with an expert on security about your particular deployment issues.

Securing your network traffic

When accessing a Web site, the content of the Web page that is returned is sent across the network in plain text. If you have a network sniffer, either a hardware device or software tools like Ethereal (http://www.ethereal.com), you can capture all the network traffic that passes through your device.
If your Web site is on the Internet, any user accessing the Internet Service Provider’s site can potentially see the data passing between your Web server and whoever is accessing it. If your Web site is an intranet, then anyone on the same physical network, in other words, most employees, can see the data.

If the data is private or confidential, then you should secure it. On an Internet site, confidential information might be credit card details or customer information; on an intranet site, it might be salary details or bank account information if you are using a Web-based HR/payroll application.

Securing this traffic is fairly easy; you must enable HTTP/S or Secure Socket Layer (SSL) for HTTP on your Web server. HTTP/S encrypts the data flowing between your Web server and the client process (normally a Web browser) using Public Key Cryptography. You will need a Digital Certificate to allow this encryption to take place.

You can purchase SSL certificates for a public Web site from Verisign (http://www.verisign.com), Thawte (http://www.thawte.com), GeoTrust (http://www.geotrust.com), and others. These sites also have documentation explaining SSL and the process for purchasing, installing, and configuring SSL on many popular Web servers.

Note: You should purchase the highest level of encryption possible for your locality. Most countries now allow 128-bit SSL, while some are still limited to 40-bit. The Digital Certificate provider will let you know the highest level that you can purchase.

If you are hosting a private Web site or an intranet, then you can generate your own certificates. This has the benefit of being free, but the users of your site will have to accept their Web browser's warning that the certificate from your site is not trusted. To generate your own certificates, see your Web server's documentation.

After you have enabled SSL, you can use https instead of http as the URL protocol for your Web site, and then the data will be encrypted. For example, if your Web site address is:

http://www.mysite.com

You can now use:

https://www.mysite.com

Securing your Web server

Since your Web server is the first computer that users access, it is also the first machine you should secure.

Caution: To ensure that your Web server is secure, install the latest security patches for the operating system and the application running the server.

For a publicly accessible Web site, you should minimize the other services running on this machine. This provides better security, as the fewer things running on this machine, the fewer things can go wrong or be compromised.

You should also read all the Web server's documentation that deals with security. Most Web servers ship with most security settings disabled. You should go through all the settings and turn off any Web server features that you do not need.
Hiding your Web server type and version

It is good practice to hide the brand and version of your Web server process to make it harder for "script-kiddies" to find out which Web server you are using.

To see how your Web server responds, use a Telnet session to access the port that the Web server is listening to. The default port is 80. The following procedure shows the commands to type. Replace the hostname with your Web server's name. You might find that when you type GET / HTTP/1.0 it might not be echoed back to you:

To check your Web server response:

1. Type telnet hostname 80 and press ENTER.
2. Type GET / HTTP/1.0 and press ENTER twice.

Note: Be sure to type a space preceding and following the first / in the GET / HTTP/1.0 command.

The following is echoed back to you:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
Date: Fri, 11 Jul 2003 16:59:53 GMT
Content-Type: text/html

... HTML text of the default page ...
```

In the previous example, you can see that the Web server is Microsoft's Internet Information Server (IIS) Version 5.1.

If you can modify the HTTP headers, make the Server setting return a generic name, like WebServer. Consult your Web server's documentation to see if it is possible and how to modify the HTTP headers.

Changing your script directory names

You should not use the standard script directory names. If you have an Apache server, do not use cgi-bin. If you are using Microsoft's IIS, do not use Scripts. See your Web server documentation for instructions on how to create a different script directory.

Most Web servers also ship with default home pages, as well as demonstration scripts. These generally should be disabled or deleted.

Hiding the CGIIP executable name from the end user

Hiding the WebSpeed Messenger name from the end user also provides a level of security. When you access a WebSpeed application, the URL used will look similar to the following if you are using Windows as the Web server:

```
```
If you are using UNIX, then it will look similar to the following:

```
http://www.mysite.com/cgi-bin/wspd_cgi.sh/WService=Orders/main.r
```

Using the default names is bad security practice because it lets people know what application server you are using, in this case WebSpeed. For example, if you perform a Google search for `wspd_cgi.sh` or `cgiip.exe`, you will find many sites using WebSpeed. Some of these are not securely deployed.

**Microsoft IIS**

If you are using Microsoft IIS, then WebSpeed includes an example file explaining how you can hide the Messenger's name. It is called `cgiip.wsc` and, by default, is located in the `C:\InetPub\Scripts` directory. It is recommended that you rename the file to something that is meaningful only to you, for example, `orders.inet`. The extension (`.inet`) must be an unused extension on your machine. You should also delete the `cgiip.exe` and `wsisa.dll` Messenger files in the `Scripts` directory.

If you open the `orders.inet` file using a text editor, you will see instructions on how to configure IIS to run this script when it is part of the URL.

**Note:** If you are using IIS 4.x or 5.x, you might find that the `Configuration` button mentioned in the instructions is disabled. To enable the `Configuration` button, first choose the `Create` button just above it.

Use the extension you have chosen (for example, `.inet`) instead of the `.wsc` extension mentioned in the instructions.

At the end of the newly created `orders.inet` file, change the WebSpeed service name from `wsbroker1`. For the example above, use `Orders`.

All lines beginning with `#` are comments. The only required line is the one that references the service name or host and port of the WebSpeed broker.

Assuming that you have changed the `Scripts` directory to be `web`, the URL becomes:

```
http://www.mysite.com/web/orders.inet/main.r
```

If you have more than one WebSpeed service, then you will need a `.inet` file for each service.

**UNIX**

There are many different Web servers available on UNIX. To find out which Web servers Progress Software Corporation has tested and certified, search the Knowledge Center. You can access the Progress Knowledge Center from the Support page at `http://www.progress.com/support/index.ssp`. 
Each of these has different configuration instructions. You should read the documentation supplied by the Web server vendor to determine how to enable CGI applications. Rename the Progress-supplied `wspd_cgi.sh` to something that is meaningful only to you and change the WebSpeed service name from `wsbroker1`. If you have changed the `cgi-bin` directory to `web` and allowed `.inet` scripts to be run as CGI programs, then the URL you would use is:

```
http://www.mysite.com/web/orders.inet/main.r
```

**Minimizing access to the WebSpeed Messenger Administration tool**

If the Messenger Administration tool is enabled, users can see your configuration information. This information can then be used to compromise your application.

To disable this feature, you can do one of two things: either disable the feature totally or allow only "trusted" IP addresses to access the Messenger Administration tool.


To allow a list of IP addresses to access the Messenger Administration tool, edit the `ubroker.properties` file on the Web server and set the `AllowMsngrCmds` to 1 (one) and the `wsmAdmIPLList` to a comma-separated list of IP addresses that are permitted to access the Messenger Administration tool. This needs to be done in the appropriate Messenger section: `[WebSpeed.Messengers.CGIIP]`, `[WebSpeed.Messengers.WSASP]`, `[WebSpeed.Messengers.WSISA]`, or `[WebSpeed.Messengers.WSNSA]`.

If the Messenger Administration tool is enabled, you can change the default WebSpeed Messenger Error Messages, as described in *OpenEdge Application Server: Developing WebSpeed Applications*.

You can also verify your WebSpeed configuration. Use the following URLs to see the Messenger Administration tool:

```
```

Or:

```
http://www.mysite.com/cgi-bin/wspd_cgi.sh?WSMAdmin
```
Securing your WebSpeed server machine

Having secured your Web server machine, you now must secure your WebSpeed server. This machine has, at least, an AdminServer and a WebSpeed server running on it. When you installed OpenEdge on this machine, you should have also enabled the AdminServer security mentioned in OpenEdge Getting Started: Installation and Configuration. You should make sure that all the vendor's security patches for this operating system have been applied, and check to see that the latest Progress Service Pack has also been installed. As you did with the Web server, you should also minimize other services running on this machine. This provides better security, as the fewer things running on this machine, the fewer things can go wrong.

The WebSpeed broker's configuration should also specify an owner. This allows the WebSpeed broker and agents to be started with the specified user's rights, not the root or system administrator's rights. See OpenEdge Application Server: Developing WebSpeed Applications for details.

You should always have a separate WebSpeed server for development/testing and production. These should also use different Web server machines and be assigned to different NameServers to reduce the chance of outside access to the development machine.

Figure 15 on page 94 shows a deployment model that uses separate machines for the Internet production, intranet production, and the development/test servers. The databases are all installed on the same machine as the WebSpeed servers. This is the preferred route if your machine has the capacity to host both, as it will provide the best performance.
Figure 15: Deployment model with separate machines for Internet Production, Intranet Production, and Development/Test servers

Figure 16 on page 95 shows a deployment model that uses two NameServers and puts all the production databases on one machine. This is useful because the intranet and Internet applications might be sharing some of the data from each database, and it lets you split the number of agents between Internet and intranet access, saving license fees.
All access from the Internet goes through the Internet NameServer, and all intranet access (both production and development/test) goes through the intranet NameServer. This means that the Internet NameServer only knows about the Internet applications and cannot hand requests to the intranet production or development/test WebSpeed servers.

Using an AppServer to run your business logic allows you to place another level of indirection between your application and the database. This enhances the security of the application, as the WebSpeed server does not directly connect to the database; it accesses the data through the AppServer. See *OpenEdge Application Server: Developing WebSpeed Applications* for information on how this can be achieved.

### Securing your WebSpeed application

To make sure that your application is as secure as possible, follow the suggestions outlined in the following sections.
Using DBAUTHKEY to lock your r-code to the database

An under-used feature of OpenEdge is the DBAUTHKEY (and RCODEKEY) features of PROUTIL.

With DBAUTHKEY, you assign a key to the database and then any code compiled against that database will have the key in it. When it comes time to run the code, if the key in the database does not match the key in the r-code, you will get an error similar to the following:

```
** CRC for table does not match CRC in program. Try recompiling. (1896)
```

If you already have r-code deployed, use the RCODEKEY function of PROUTIL to tag the existing r-code without the must recompile.

See OpenEdge Data Management: Database Administration for more information on using the DBAUTHKEY and RCODEKEY features of PROUTIL.

Use the agent's production setting

For production environments, either Internet or intranet, you should set the agent application mode to Production. You can set this option through OpenEdge Management or OpenEdge Explorer. After setting this, you should stop and start the WebSpeed service to activate the change.

Modifying web-disp.p

By default, the agents run web/objects/web-disp.p as their startup program. Each request that is issued to an agent runs through this code. This is the best place to control what happens to each request.

Modify web-disp.p to:

- Make sure that certain r-code can only be run by certain users
- Turn off the PING or DEBUG facilities
- Connect to a database every time a request comes through
- Check for a user timing out

Because each request must go through this code, any changes made to web-disp.p are system wide.

If you want to change this code, you should move it into your application's source tree and rename it. This way, when a service pack installs a newer version of web-disp.p, your changes are not overwritten. You should also compare your code with the new code shipped in the service pack to make sure you also incorporate any bug fixes or enhancements.

The following shows a simplified version of the default WebSpeed web-disp.p.
Default web-disp.p

/* Set the web-request trigger. */
ON "WEB-NOTIFY":U ANYWHERE DO:
  OUTPUT {&WEBSTREAM} TO "WEB":U.
/* Parse the request/CGI from the web server. */
RUN init-cgi IN web-utilities-hdl.
/* Initialize for web-request. */
RUN init-request IN web-utilities-hdl.
AppProgram = (IF AppProgram = "debug":U THEN "webutil/debug.p":U ELSE
  (IF AppProgram = "ping":U THEN "webutil/ping.p":U ELSE
   (IF AppProgram = "reset":U THEN "webutil/reset.p":U ELSE
    AppProgram)))
RUN run-web-object IN web-utilities-hdl (AppProgram) NO-ERROR.
/* Run clean up and maintenance code */
RUN end-request IN web-utilities-hdl NO-ERROR.
/* Output any pending messages queued up by queue-message() */
IF available-messages(?) THEN
  output-messages("all", ?, "Messages:").
OUTPUT {&WEBSTREAM} CLOSE.
END. /* ON "WEB-NOTIFY" */

/* Wait for a web-request to come in */
WAIT-FOR-BLOCK:
REPEAT ON ERROR UNDO WAIT-FOR-BLOCK, LEAVE WAIT-FOR-BLOCK
  ON QUIT UNDO WAIT-FOR-BLOCK, NEXT WAIT-FOR-BLOCK:
    WAIT-FOR "WEB-NOTIFY":U OF DEFAULT-WINDOW.
END. /* WAIT-FOR-BLOCK: REPEAT... */

Note: Default web-disp.p and Secure web-disp.p do not run. Much of the code has been removed. The purpose of these examples is to show program flow.

Secure web-disp.p shows a simplified, secure web-disp.p. You insert the bold text into the original web-disp.p replacing the "AppProgram = ..." code.

This code stops PING, DEBUG, and RESET, changes the extension of any requested program into r-code, checks that the r-code file exists, and verifies if this r-code is valid for this user by looking up a database table called UserPrograms. You must create a table called UserPrograms containing (at least) both these fields. Also, UserID is a variable that you must instantiate.

You usually use a cookie, hidden fields, or URL parameters to hold the user's ID. This should be encrypted in a suitable manner. See Parameter passing on page 98 for an example of encrypting this ID.
Secure web-disp.p

/* Set the web-request trigger. */
ON "WEB-NOTIFY":U ANYWHERE DO:
   DEFINE VARIABLE vLocn AS INTEGER NO-UNDO.
   OUTPUT {&WEBSTREAM} TO "WEB":U.
   /* Parse the request/CGI from the web server. */
   RUN init-cgi IN web-utilities-hdl.
   /* Initialize for web-request. */
   RUN init-request IN web-utilities-hdl.
   /* Remove current extension */
   vLocn = R-INDEX (AppProgram, ".").
   IF vLocn > 0 THEN
      AppProgram = SUBSTR (AppProgram, 1, vLocn - 1).
   /* Add a .r */
   AppProgram = AppProgram + ".r".
   /* Can this User run this program OR does it exist? */
   IF NOT CAN-FIND (UserPrograms WHERE UserPrograms.UserID = UserID
      AND UserPrograms.Program = AppProgram)
      OR SEARCH (AppProgram) = ? THEN
      AppProgram = "NotValidProgram.r".
   RUN run-web-object IN web-utilities-hdl (AppProgram) NO-ERROR.
   /* Run clean up and maintenance code */
   RUN end-request IN web-utilities-hdl NO-ERROR.
   /* Output any pending messages queued up by queue-message() */
   IF available-messages(?) THEN
      output-messages("all", ?, "Messages: ").
      OUTPUT {&WEBSTREAM} CLOSE.
   END. /* ON "WEB-NOTIFY" */

After creating your new-web-disp.p, you must change the agent parameters to reference it.

Minimize the PROPATH

It is essential that the PROPATH is kept to a minimum, both for performance and security. The OpenEdge-install/tty directory and all the r-code libraries (*.PL) in the OpenEdge-install/tty directory are added to the end of your PROPATH setting by default. This means that there are many programs in your PROPATH that you did not write and anyone can run these programs by adding them to the end of your URL.

To avoid this, simply rename the OpenEdge-install/tty directory to OpenEdge-install/tty_save. Then, copy all the r-code files you use to a new directory called tty in your deployment area and add this to the end of your PROPATH. Remember that some of the r-code files WebSpeed might use are in the .PL files, and you must extract them using the PROLIB utility documented in OpenEdge Deployment: Managing ABL Applications.

Parameter passing

If you want to pass parameters between Web requests, you can use hidden fields on forms, URL parameters, cookies, or a combination of each technique. Each technique has pros and cons. Hidden fields only work on forms, URL parameters are visible to the end user, and cookies are not allowed by some users.
The simplest way to pass many parameters between Web requests is to use the database. You pass a unique identifier for each user or session between requests, and use this as a key into a "state" table held in the database. This technique requires that only a small token be passed between requests, as the majority of the data is safe and secure in the database.

Do not pass the unique identifier in plain text. Doing so makes it very easy for an end-user to change the value (even in hidden fields or cookies) and become someone else. Use code, similar to the code shown below to prevent people from changing the unique identifier, unless they know the hidden words, in this case "Web" and "Speed."

**Passing unique identifiers**

```plaintext
/* This code assumes that the Unique ID will not contain any colons (:). */
DEFINE VARIABLE vToken AS CHARACTER NO-UNDO.
DEFINE VARIABLE vUniqueID AS CHARACTER NO-UNDO.
/* WebEncode function */
FUNCTION WebEncode RETURNS CHARACTER (pUniqueID AS CHARACTER):
  RETURN pUniqueID + ":" + ENCODE ("Web" + pUniqueID + "Speed").
END.
/* Use this to encode the Unique ID, then pass as parameter */
vToken = WebEncode (vUniqueID).
/* Use this to decode the token passed as a parameter. */
vUniqueID = ENTRY (1, vToken, ":").
IF vToken = WebEncode (vUniqueID) THEN
  /* vToken has not been modified */
ELSE
  /* ERROR - vToken has been modified */
```

**Firewalls**

A firewall is the first line of defense for basic network security. It is usually a separate device that sits between the untrusted network (the Internet) and the trusted network (the intranet). The role of a firewall is to stop unauthorized access of information in the trusted network by individuals on the untrusted network, but allow defined access from the trusted to the untrusted.

An analogy for a firewall is a moat around a castle with the drawbridge being the firewall device. The drawbridge is controlled by guards who only allow certain traffic in, usually after inspecting it, and will allow outbound traffic if it has permission.

There is usually a third network commonly referred to as the DMZ or Demilitarized Zone. This network is separate from both the others, but it can communicate with both. This is a semitrusted area that is protected by the firewall, so only certain traffic can come in. Any traffic coming from the DMZ into the trusted network must abide by strict rules, so errant requests are denied. There are three physical network ports on a DMZ-enabled firewall, one for each network.

*Figure 17* on page 100 shows a firewall with a DMZ. This is the usual configuration for a firewall.
Figure 17: Firewall with DMZ

Figure 18 on page 101 shows a more secure firewall configuration. The reason for having two firewall devices from different manufacturers is two-fold. First, having only one device means that any bugs or security holes in the firewall software could allow direct connection between the untrusted and trusted networks. Second, using different manufacturers’ hardware/software combinations stops hackers from using the same exploit or security hole on both devices.
Firewalls can be implemented in either hardware or software. A hardware firewall is a machine that has a proprietary operating system and software for providing the service. Any patches provided by the firewall supplier should be applied as soon as possible to minimize the risk of attack.

A software firewall is a program that is loaded onto a general-purpose computer, usually a PC, to provide the service. To be effective, software firewalls rely on the underlying operating system to be secure, so you should make sure that all the operating system manufacturer's patches are applied along with any updates to the firewall software. You should avoid running anything else on a software firewall's host machine. Some software firewalls do not use an underlying general purpose operating system; they use standard hardware, but load their own proprietary operating system along with the firewall software.

For more information on setting up firewalls, see Firewall configuration and debugging on page 102.
Firewall configuration and debugging

If you are deploying a public WebSpeed application or a Progress AppServer™ Internet Adapter-enabled AppServer application, then you should be using firewalls to minimize the risk of network intrusions.

The following sections explain how to configure a firewall to allow WebSpeed to function and how to debug a nonworking firewall deployment.

Firewall configuration

Using a firewall poses additional configuration issues because you must configure the firewall to allow communications between the OpenEdge server host machines on particular ports using TCP or UDP protocols. In OpenEdge Application Server: Developing WebSpeed Applications, the entire round-trip request is shown. All of these messages must go through the firewall.

The following figure illustrates which ports must be open and what protocols the messages use.

**Figure 19: Firewall configuration**
Note: The firewall fire2 has two network cards, one for the Demilitarized Zone (DMZ) and one for the Internal network. Each of these has its own IP address, as shown.

In the following sections the hosts file is mentioned. On UNIX or Linux, this is located at /etc/hosts and in Windows NT and later at C:\WINDOWS\system32\drivers\etc\hosts.

The WebSpeed Messenger ubroker.properties file must have the minNSClientPort and maxNSClientPort settings modified in the [WebSpeed.Messengers] configuration, as shown below. The port range must be big enough to cope with all the potential simultaneous requests from the Internet. In this case, there are 20 ports available. You can make this range bigger if needed. Also, you must change the setting for the NameServer to point to the correct host.

Configuring ubroker.properties file for firewall

```
[WebSpeed.Messengers]
  ...
  minNSClientPort=5680
  maxNSClientPort=5699
  controllingNameServer=InternetNS
  ...
[NameServer.InternetNS]
  ...
  hostName=inet_ns
  location=remote
  portNumber=5678
  ...
```

You must configure the following:

- Between the Internet and Web server webserv1, allow inbound and outbound traffic from the Internet on port 80 (the default for HTTP) to the Web server. This is a standard configuration on most firewalls. If you are using HTTP/S (HTTP over SSL), then the default port is 443.

- Between the Web server (Messenger) and NameServer, allow:
  - UDP from IP Address 1.1.1.1 to 5.5.5.5 on port 5678. This is the inbound NameServer request traffic.
  - UDP from IP Address 5.5.5.5 to 1.1.1.1 on ports 5680 to 5699 inclusive (assuming the above settings in the ubroker.properties file). This is the outbound NameServer response traffic as specified in the Messenger's ubroker.properties file on the Web server.

- Between the Web server (Messenger) and WebSpeed broker, allow:
  - TCP from IP Address 1.1.1.1 to 4.4.4.4 on port 7800 for the inbound request.
  - TCP from IP Address 4.4.4.4 to 1.1.1.1 on port 7800 for the outbound reply.

- Between the Web server (Messenger) and WebSpeed agents, allow:
  - TCP from IP Address 1.1.1.1 to 4.4.4.4 on ports 7801 to 7805 inclusive for the inbound request.
  - TCP from IP Address 4.4.4.4 to 1.1.1.1 on port 7801 to 7805 inclusive for the outbound reply.
Most firewalls accomplish this by using "port forwarding." This means that when the firewall receives a request from a host on a certain port in the DMZ, it is passed through to a particular host on the internal network. When the webserv1 machine makes a request to the NameServer, it cannot see IP address 5.5.5.5 directly, and it has to pass the request to the firewall machine fire2. The firewall then makes the request on the internal network to IP address 5.5.5.5 on its behalf. When the response comes back from the NameServer to the firewall, the firewall will send it on to the Messenger on the DMZ network. As an analogy, think of the firewall as a language interpreter where the WebSpeed Messenger speaks English and the NameServer speaks German. The Messenger needs to talk to the NameServer but cannot do so directly, so it forwards the request to the interpreter who, in turn, makes a request to the NameServer on the Messenger's behalf. The response is given to the interpreter by the NameServer, who then forwards it to the Messenger.

This is achieved by setting the hosts file on webserv1 to have the host inet_ns set to 2.2.2.2, as shown below. When the Messenger looks for host inet_ns, it uses the IP address 2.2.2.2, which is the firewall host fire2, as shown:

```
127.0.0.1   localhost
2.2.2.2   inet_ns
```

**Note:** You do not must have an entry for fire2 in the hosts file as the DMZ machines never communicate with it by name; DMZ machines believe that communication with other machines never travels beyond the internal network.

Similarly, the Messenger cannot communicate directly to the WebSpeed server host webspeed1 at IP address 4.4.4.4 either. So, another entry needs to be made in the hosts file to make the Messenger communicate with the firewall instead of the "real" host, as shown:

```
127.0.0.1   localhost
2.2.2.2   inet_ns webspeed1
```

Because of this, you cannot use the default setting for the WebSpeed broker's registration mode. The default is to use the broker host IP Address. If you do this, the NameServer will tell the Messenger to try to contact the broker on IP address 4.4.4.4, which is not a valid IP address in the DMZ, and it will appear as if the broker has not responded. You must set the broker to register using a defined host name, in this case webspeed1. When the NameServer responds this time, it tells the Messenger to try to connect using the host name webspeed1. The Messenger asks the operating system on its host for the IP address of webspeed1. Since the address set in the hosts file is 2.2.2.2, the Messenger uses that address when it is returned. The firewall then gets the request and passes it through.

**Note:** The NameServer and WebSpeed server hosts do not need the firewall IP address in their hosts file because they only respond to requests and do not make them.

**Debugging firewall configurations**

After configuring the firewall, you must test the configuration to see if it works. The easiest way to do this is to try to run the WebSpeed application from the Internet. This probably means you must disconnect the test client PC from the internal network and then dial an Internet Service Provider (ISP) to then act as a "real" Internet client.
First, make sure everything works by entering the URL for the application into your Web browser. In most cases this method fails because the firewall configuration omitted one or two ports or a `ubroker.properties` setting was left unchanged.

For help in tracing the cause of the failure, see WebSpeed request round-trip on page 23 to remind yourself what the entire round-trip process is and test each stage one at a time. The error shown by the Messenger (if it worked that far) will lead you to the answer as well.

**Note:** You might want to use a software tool like Ethereal (http://www.ethereal.com) to allow you to see what packets are traversing the network.

If you are using Microsoft Windows 2000 or later to host the Web server, you might find that UDP or TCP packets are being sent, but they are being ignored by the Web server machine. This can be caused by incorrectly setting the IP Packet filter. All ports used for the firewall access must be allowed in the IP Packet filter. Packet filter settings are addressed in the following topics.

To access the IP Packet Filter settings:

1. In the **Windows Control Panel**, click **Network Connections**.
2. Right-click on your LAN connection and select **Properties** from the pop-up menu. The **Local Area Connections Properties** dialog box appears:

![Local Area Connection Properties](image)

3. Choose **Properties**. The **Internet Protocol (TCP/IP) Properties** dialog box appears:
4. Click **Advanced**. The **Advanced TCP/IP Settings** dialog box appears:

5. Choose the **Options** tab, as shown:
6. Highlight **TCP/IP Filtering** in the list and then click **Properties**. The **TCP/IP Filtering** dialog box appears.

7. You can set the filter to allow all packets as shown, or you can restrict the ports allowed by adding them into the appropriate areas:

---

**Note:** If you must use DNS, then you also must allow UDP port 53 and TCP port 53. For the Web server, you need port 80. For HTTP/S, you need port 443.
Web server access

Can your Web browser access the Web server? Put a test HTML file in the Web server’s root directory to see if you can access it with http://webserver/test.htm. If you can, then delete the test HTML file and move on. If the file does not appear, check to see if the Web server is running.

WebSpeed Messenger

Does the WebSpeed Messenger run? If you get a “WebSpeed error from messenger process (6019)” error message, then the WebSpeed Messenger is running. If not, you should enable the Messenger logging function in ubroker.properties as shown in the excerpt below. The default logging level is 1, which is Errors Only. The following setting should show the issues:

```
[WebSpeed.Messengers]
... 
  logFile=@{WorkPath}\msgr.log
  loggingLevel=1
... 
```

You might have received a Web server internal error. This is usually caused by a Web server misconfiguration related to the “executability” settings for CGI programs. Check your Web server documentation to make sure you have configured it to run your WebSpeed Messenger correctly.

If you use the Messenger Administration tool, you can test the configuration of your WebSpeed application. For more information, see Configuring a WebSpeed Messenger.

NameServer Access

If the Messenger is working, the next step is to confirm that the NameServer is being accessed. Set the NameServer’s logging level to 3 (Verbose) using OpenEdge Management/OpenEdge Explorer or by editing ubroker.properties manually. To enable this change, you must stop and restart the NameServer and wait for the WebSpeed broker to inform the NameServer that it is available.

If the NameServer does not know about a service, it cannot direct clients to it. To check the NameServer to see if it knows about the WebSpeed server, you can use OpenEdge Management/OpenEdge Explorer or you can use the code NSMAN -name NS1 -query, as shown in below.
Checking NameServer access using NSMAN -name NS1 -query

<table>
<thead>
<tr>
<th>Application Service</th>
<th>UUID</th>
<th>Name</th>
<th>Host</th>
</tr>
</thead>
</table>

Now, try to access the application again. After the error is returned to the WebSpeed Messenger, check the NameServer's log file. You should see something similar to the following:

```
Thread-0>(26-Jul-03 18:42:15:107) Request received from 192.168.123.110 2167 for WS.Sports2000_WS. (8201)
Number Of Brokers = 1. (8206)
Thread-0>(26-Jul-03 18:42:15:107) Response sent to 192.168.123.110
```

If you do not see the "Request received" in the log file, then the firewall is losing the inbound NameServer request. Otherwise, the outbound request is being lost. After debugging this stage, make sure to reset the logging setting.

**Accessing the WebSpeed broker**

This time, set the WebSpeed broker's logging setting to `verbose` and make the request. You should see something similar to the following (at a time just after the NameServer log entry):

```
L-3055>(26-Jul-03 18:57:53:816) Received connection:: (8125)
C-0001>(26-Jul-03 18:57:53:836) Client connected : . (8533)
C-0001>ubWSclientThread.processConnRsp(): ubRsp = 0, getNeedNewConnID() = false
C-0001>(26-Jul-03 18:57:53:836) The client C-0001 has disconnected from the broker. (8084)
C-0001>(26-Jul-03 18:57:53:836) Client disconnected : . (8534)
```

If the WebSpeed Messenger error says it could not contact the broker, but the broker log file says it was contacted, then the fault is on the return path. If there is no contact logged in the broker log file, then it did not receive the message. Either of these will point to the firewall rule that was left out or misconfigured.

**Accessing the WebSpeed agent**

Use the ABL Trace function to see if the agent received the request. Configuring this feature is covered in *OpenEdge Deployment: Managing ABL Applications.*
General notes on debugging

Think through the request process and see what the error messages say. This will lead to the issue most of the time.

Check the log files of the firewall itself. These will show what messages are flowing through it. You will probably have to filter these because a production firewall will have more than just your WebSpeed requests going through it.

Always use a new Web browser window for each test request. Most browsers attempt to speed up requests by caching information. A subsequent test in the same browser window can return cached data instead of the proper results for the new settings. This can also be achieved by using the "Reload" function of the browser. See your browser documentation for information on setting your browser to not use cached copies of pages.

Optimizing WebSpeed performance

This section discusses the following WebSpeed performance issues:

- How requests affect performance on page 110
- Browser (HTTP) response times on page 111
- HTTP/S performance on page 111
- Using different Messengers on page 111
- Multiple Web servers on page 112
- Domain Name System on page 112
- Multi-homed servers (multiple IP address servers) on page 112

How requests affect performance

In WebSpeed request round-trip on page 23, there are quite a few steps that make up the entire round-trip process and possibly quite a few separate machines, ranging from the Web server, firewalls, NameServer, WebSpeed server (broker and agents), and probably a database as well. Each of these steps introduces performance challenges of their own. The Web server must cope with not only the WebSpeed requests, but the normal HTML requests as well. Firewalls can introduce network latency, as some will inspect each packet to make sure it is "allowed" before passing it through to the next machine in the process. The NameServer performance issues are covered in the previous section.

The WebSpeed broker and agents have a role to play in performance as well. The broker launches and configures new WebSpeed agents before they are needed. This enables the requests that are being received to wait as short a time as possible in the broker's request queue. The launching of a new agent will take a period of time—the agent itself needs to be loaded into memory, possibly run some application code to create super procedures, and connect itself to the database. To keep free agents, you should set the Minimum agents to a number higher than 0. This setting controls how many agents the broker will keep free, up to the Maximum agents.
Consider the case of a broker that will start five agents. It will keep at least two free agents at all times unless it has already launched the maximum number of agents, which is 10. Agents that are not used for a period of time will be eliminated. The **Auto Trim Timeout** setting in the broker's **Advanced Features** tab controls this time period and is entered as a number of seconds, so the default of 1800 is equivalent to 30 minutes.

For more information, see *OpenEdge Management and OpenEdge Explorer: Configuration* or the OpenEdge Management or OpenEdge Explorer online help.

---

**Browser (HTTP) response times**

Having a good response time for a Web site is very important. Making sure that the Web server is configured well and has enough memory and CPU performance is important in providing good response times.

Hosting static images and HTML on the Web server allows the Web server to cache these and provide better performance.

---

**HTTP/S performance**

Using HTTP/S to encrypt the traffic between the Web browser and the Web server provides very good security for the data, but it also introduces performance degradation. In general, for each request that is made to the Web server, the Web browser and Web server must go through at least 8 and up to 13 handshake messages before the actual data is sent. Also, one of these handshake messages needs the Web browser to generate a long random number, which is a slow process.

Because a Web page is usually made up of multiple requests, using HTTP/S as the protocol slows down the Web page being displayed. For example, a Web page with 15 images on it will mean 16 individual requests to be made to the Web server.

HTTP/1.1 has features that should allow one connection with multiple requests to work, but the implementation into the Web servers and the Web browsers has not been completed. There are hardware SSL accelerators on the market that will alleviate most of the performance issues on the Web server side when using SSL.

---

**Using different Messengers**

Throughout this manual, the WebSpeed Messenger that has been described is **cgiip** or **cgiip.exe**. Depending on your Web server, there are alternate WebSpeed Messengers. If you are using a Microsoft IIS Web server, you can use the **wsisa.dll** Messenger, and if you are using the Netscape/iPlanet Web server, which is now part of SunOne, you can use the **nsapi.dll**.

Each of these Messengers acts in exactly the same way as **cgiip**, but because they are Dynamic Link Libraries (DLLs), they stay in memory and are faster to execute the next time they are called.

Being a DLL does have a drawback. If the Web server gets confused, there is a strong possibility that the Messenger process will stop working. The only way to correct this is to restart the Web server process on the Web server machine; sometimes this involves a reboot.

Because the **cgiip** Messenger being loaded each time a request is made, it is slower than the DLL versions, but it is also more reliable. Since the time it takes the **cgiip** Messenger to load itself into memory is quite small, using **cgiip** is a good idea for production Web sites, as the performance overhead is slight, but the reliability is high. You should test each possible Messenger for performance and determine which one you want to use. During testing, remember to time the entire application, not just the Messenger load times.
Multiple Web servers

One way to increase the throughput of the Web server is to have more than one and share the load. This is easily achieved using WebSpeed, because the Messenger configuration is identical on each Web server. To make more than one Web server respond to requests for the same Web site, you can use DNS round-robin aliases or a hardware redirection. For more information, see your router or DNS documentation.

Domain Name System

The Domain Name System (DNS) is one of the very useful, but often misconfigured, parts of any IP network. The job of the DNS is to resolve host names into IP addresses (and occasionally the reverse). If you misconfigure the DNS, then you will get some odd problems. The most common is that the client is taking a very long time to connect to the application server. This is probably due to a DNS issue where the DNS lookup is going outside your local LAN and onto your Internet Service Provider’s DNS servers.

If you are experiencing any problems at connect time, it is very important to check that your DNS is configured correctly. To do this, go to a prompt (either UNIX or a Windows Command prompt) and enter `nslookup servername`. The time it takes to respond is the DNS lookup time, and it should be nearly instantaneous.

Note: If you are using a `hosts` file entry to “fix” this problem, then you really should fix the DNS problem and remove the entry in the `hosts` file.

Multi-homed servers (multiple IP address servers)

If you have a server that has more than one network card in it, it is probably going to have more than one IP address—a scenario described as multi-homed servers. You could have problems connecting to the AdminServer and other OpenEdge servers in this instance.

An example of a multi-homed server is shown in the following figure.
The clients on the 10.x.x.x subnet will not be able to access services on the host unless they use the correct IP address 10.1.1.1. Likewise, the 192.168.x.x subnet must use 192.168.123.1. To make all these clients connect to a single AppServer on the host, set up the `hosts` file on the machines in the 10.x.x.x subnet to read as follows:

```
127.0.0.1 localhost
10.1.1.1 hydra
```

And, on the 192.168.x.x subnet to read as follows:

```
127.0.0.1 localhost
192.168.123.1 hydra
```

All AppServer and WebSpeed instances running on `hydra` must have their **Register with NameServer** settings changed so that they do not register with an IP address, but instead with a host name. You can set these properties in OpenEdge Management or OpenEdge Explorer from within the configuration of each AppServer broker or WebSpeed broker. For more information, see *OpenEdge Management and OpenEdge Explorer: Configuration* or the OpenEdge Management or OpenEdge Explorer online help.

This enables the NameServer to tell the clients, regardless of what subnet they are on, to connect to the machine called `hydra`. It is then up to the client machine to decide what the appropriate IP address is.
Running sample applications

WebSpeed includes sample applications that illustrate intranet, Internet, and extranet applications. The sample applications run against the sample Sports2000 database, which is located in your installation directory.

Note: For detailed instructions on creating a local copy of the Sports2000 database for use with a sample application, see the chapter on working with sample applications in OpenEdge Getting Started: Progress OpenEdge Studio.

The HTML source files that are compiled to create the Web objects (r-code) for the sample applications are in the install-path/src/samples/web directory. The r-code that WebSpeed agents actually run is in the install-path/tty/samples/web directory.

Before you can run sample applications, you must have WebSpeed running on a Web server. In addition, you must set up WebSpeed to run while connected to the Sports2000 sample database.

To set up WebSpeed to run while connected to Sports2000:

2. Copy the sports2000trgs folder (which contains database triggers) from your OpenEdge installation directory to your working directory.
4. Start a WebSpeed broker connected to your working copy of the Sports2000 database. See Configuring a WebSpeed broker to connect to a database on page 116 for instructions on completing this procedure.

Overview of sample applications

Once you have the broker and database running, you can run the following sample applications in a Web browser:

- **SportsPro Shopping Cart internet application** — An example of one of the most popular types of Web-based, business-to-consumer applications.

  To run SportsPro Shopping Cart, enter a URL using the following format:

  http://host_name/scripts_dir/messenger/WService=wsbroker1/samples/web/internet/home.r

- **Supplier extranet application** — An example of a business-to-business extranet application with some advanced WebSpeed features, including a login dialog box.

  To run the Supplier application, enter a URL using the following format:

  http://host_name/scripts_dir/messenger/WService=wsbroker1/samples/web/extranet/login.r
• **Sales Advisor intranet application** — An example of an internal business application that allows to query and update the Sports2000 database.

To run Sales Advisor, enter a URL using the following format:

```
http://host_name/scripts_dir/messenger/WService=wsbroker1/samples
/web/intranet/advisor.r
```

For example, if you are running a local IIS Web server and using the CGIIP Messenger, the URL would look similar to the following:

```
http://localhost/scripts/cgiip.exe/WService=wsbroker1/samples/web
/intranet/advisor.r
```

---

**Running the SportsPro Sales Advisor**

This section contains more information about running the SportsPro Sales Advisor application. It also explains some of the features of the application. SportsPro Sales Advisor is an intranet application that supplies helpful information to sales reps of the fictional SportsPro sporting goods company. The SportsPro Sales Advisor allows sales reps to check inventory and maintain customer account information in the Sports2000 database, directly from their Web browsers.

To run SportsPro Sales Advisor:

1. Complete the setup procedures described in Running sample applications on page 114 and make sure that WebSpeed is running in conjunction with a Web server and the Sports2000 sample database.
2. Start a Web browser (usually Netscape or Internet Explorer).
3. Enter a WebSpeed URL where `Wservice` is set to:

```
http://host_name/scripts_dir/messenger/WService=wsbroker1/samples/web/
intranet/advisor.r
```

For example, if you are running a local IIS Web server, using the CGIIP Messenger and the default broker, the URL would look similar to the following:

```
http://localhost/scripts/cgiip.exe/WService=wsbroker1/samples/web/
intranet/advisor.r
```

The SportsPro Sales Advisor application launches in your Web browser and displays an initial frame set that contains three separate Web objects: an application header (SportsPro Sales Advisor), an application index (the **Inventory Info.** and the **Customer Info.** links), and a welcome screen (**Welcome to SportsPro Sales Advisor**). The initial frame set looks similar to the following:
Configuring a WebSpeed broker to connect to a database

This section describes how to use OpenEdge Management or OpenEdge Explorer to configure and start a WebSpeed broker with agents connected to your working copy of the Sports2000 database. Before you begin, you should have a database server started for your working copy of the database.

**Note:** The following procedure assumes that you have started OpenEdge Management or OpenEdge Explorer and have connected to the appropriate host machine.

To start a WebSpeed broker and WebSpeed agents in OpenEdge Management or OpenEdge Explorer:

1. Click **Resources** in the management console menu bar. Expand the container name; the main resource types appear in the list frame.
2. In the list frame, expand the **OpenEdge** category.
3. Expand the **WebSpeed** category, and select a WebSpeed Transaction Server Broker. The **Details** page for the broker appears in the detail frame.
4. In the **Command and control** section of the page, click **Configuration**. The **WebSpeed Configuration** page opens with that Broker's properties in focus.
5. Click the **Agent** tab.
6. Click **Edit**.
7. Add the pathname of your working copy of the Sports2000 database to **Server startup parameters** using the `-db` option; for example, `-db C:\WRK\sports2000`.

**Note:** In this example, the database is located on the same machine as the WebSpeed broker with a shared memory connection. If the server for the database is running on a remote machine (or you want to use client/server mode), you must specify the host name (using the `-H` parameter) of the database's machine and the service name (using the `-S` parameter). For more information about client/server connections, see *OpenEdge Application Server: Developing AppServer Applications*. 
8. Click **Save**.
9. Select **wsbroker1** and choose **Broker Control**.
10. Click **Start WebSpeed**. After the WebSpeed broker starts, you can monitor its performance by clicking the **Status** link on the broker's Details page.
WebSpeed Configuration and Management Utilities

This appendix explains the use of the WebSpeed configuration and management utilities. For details, see the following topics:

- Utility command syntax
- WSCONFIG utility
- WTBMAN utility

Utility command syntax

The following figure shows the conventions used in utility command syntax.

```
UNIX and Windows

<table>
<thead>
<tr>
<th>command</th>
<th>db-name</th>
<th>[ qualifier ]</th>
<th>[ parameter value ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>probkup</td>
<td>sports</td>
<td>incremental</td>
<td>-vs 708</td>
</tr>
</tbody>
</table>
```
Utility syntax conventions
The table below describes each of the command components.

Table 9: Command components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>Executable</td>
</tr>
<tr>
<td>db-name</td>
<td>Database name</td>
</tr>
<tr>
<td>qualifier</td>
<td>Additional command specification</td>
</tr>
<tr>
<td>parameter</td>
<td>Operating criteria for the command</td>
</tr>
<tr>
<td>value</td>
<td>Numeric value or file specification</td>
</tr>
</tbody>
</table>

Note: Enter parameters for UNIX and Windows exactly as shown in the syntax descriptions.

WSCONFIG utility

Use the WSCONFIG utility to help validate existing WebSpeed Transaction Server or WebSpeed Messenger configurations defined in a properties file, such as the ubroker.properties file. This utility displays the property settings associated with a WebSpeed Transaction Server or Messenger configuration, and checks that the syntax and values are valid.
The WSCONFIG utility runs locally on the machine where the WebSpeed components that you want to check are running. Because the utility does not run across the network and no AdminServer is installed during a Messenger-only install, you cannot use the WSCONFIG utility to check a Messenger-only install. Table shows the WSCONFIG utility’s syntax.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
</table>

**Parameters**

- **-name component-name**

  Specifies the name of an existing WebSpeed Transaction Server or Messenger configuration to examine. The name must match the name of an existing WebSpeed Transaction Server configuration defined in the specified properties file. Although you must specify a Transaction Server, you need not specify a Messenger. If you do not specify any name, the WSCONFIG utility analyzes all the WebSpeed Transaction Server and Messenger configurations defined in the properties file specified by the -propfile parameter.

- **-propfile path-to-properties-file**

  Specifies a filename or pathname to a file that contains the property settings to be validated, for example test.properties. If a filename or pathname is not specified, it defaults to the installation version of the ubroker.properties file (install-path/properties/ubroker.properties).

- **-messenger**

  Displays one or all of the Messengers for you to view. If -name refers to a Messenger and the -messenger parameter is used, then information appears for that one Messenger. If -name does not refer to a Messenger and the -messenger parameter is used, then information appears for all the Messengers. The Messenger names in Windows are CGIIP, WSISA, WSNSA, and WSASP. The Messenger names on UNIX are CGIIP and WSNSA.

- **-validate**

  Displays command-line help.
-help

Checks the syntax and values of property settings defined in the specified properties file.

Examples
The following table shows several examples that use the **WSCONFIG** command. Assume the Transaction Server name is **wsbroker1**.
### Table 10: WSCONFIG command examples

<table>
<thead>
<tr>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>View a Transaction Server configuration</td>
<td><code>wsconfig -name wsbroker1</code></td>
</tr>
<tr>
<td>View a messenger configuration</td>
<td><code>wsconfig -name CGIIP -messenger</code></td>
</tr>
<tr>
<td>View all messenger configurations</td>
<td><code>wsconfig -messenger</code></td>
</tr>
<tr>
<td>Validate the syntax and view the configuration of all messengers</td>
<td><code>wsconfig -propfile g:\other.properties -validate</code></td>
</tr>
<tr>
<td>defined within a different property file</td>
<td></td>
</tr>
</tbody>
</table>

---

**WTBMAN utility**

Use the **WTBMAN** utility to control the operation of a configured WebSpeed Transaction Server. The utility allows you to start a Transaction Server, query its status, start and stop additional WebSpeed Agents, trim by a certain number of agents, and shut down the Transaction Server. Table shows the **WTBMAN** utility’s syntax.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX Windows</td>
<td><code>wsconfig</code></td>
</tr>
</tbody>
</table>
### Parameters

**-name transaction-server-name**

Specifies the name of a Transaction Server.

**-kill**

Stops and removes the Transaction Server from memory, no matter what it is doing.

**-start**

Starts the Transaction Server.

**-stop**

Stops the Transaction Server.

**-query**

Queries the Transaction Server for its status.

**-addagents number-to-start**

Specifies the number of additional agents to start.
-trimagents number-to-trim

Specifies the number of additional agents to trim.

-host host-name

Specifies the name of the machine where the AdminServer is running. If a host name is not specified, it defaults to the local host name.

-user user-name

Specifies a user name and prompts for a password when logging into a remote machine. A user name and password are required only when you use the -host parameter and specify a remote host name. If you specify a remote host name with the -host parameter but do not specify a user name with the -user parameter, you receive a prompt for a user name and password.

When you specify a user name with the -user parameter, Windows supports three different formats:

• A user name as a simple text string, such as mary, implies a local user whose user account is defined on the local server, which is the same machine that runs the AdminServer.

• A user name as an explicit local user name, in which the user account is defined on the same machine that runs the AdminServer except the user name explicitly references the local machine domain, for example .\mary.

• A user name as a user account on a specific domain. The general format is Domain\User, in which the User is a valid user account defined within the domain, and the Domain is any valid server, including the one where the AdminServer is running.

-port port-number

Specifies the port number of the machine on which the AdminServer controlling the WebSpeed Transaction Server is running. If a port number is not specified, it defaults to 20931.

-help

Displays command-line help.

Examples

The following table shows several examples that use the wtbman command. Assume that the Transaction Server name is wsbroker1; the user name is tom; and the AdminServer is on the remote host finance at port 9999.
Table 11: WTBMAN command examples

<table>
<thead>
<tr>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start a local Transaction Server</td>
<td><code>wtbman -name wsbroker1 -start</code></td>
</tr>
<tr>
<td>Start a remote Transaction Server ^2</td>
<td><code>wtbman -name wsbroker1 -host finance -port 9999 -user tom -start</code></td>
</tr>
<tr>
<td>Query a local Transaction Server</td>
<td><code>wtbman -name wsbroker1 -query</code></td>
</tr>
<tr>
<td>Query a remote Transaction Server a</td>
<td><code>wtbman -name wsbroker1 -host finance -port 9999 -user tom -query</code></td>
</tr>
<tr>
<td>Add agents (for example, 2) to a local Transaction Server</td>
<td><code>wtbman -name wsbroker1 -addagents 2</code></td>
</tr>
<tr>
<td>Add agents (for example, 2) to a remote Transaction Server a</td>
<td><code>wtbman -name wsbroker1 -host finance -port 9999 -user tom -addagents 2</code></td>
</tr>
<tr>
<td>Trim agents (for example, 3) from a local Transaction Server</td>
<td><code>wtbman -name wsbroker1 -trimagents 3</code></td>
</tr>
<tr>
<td>Trim agents (for example, 3) from a remote Transaction Server a</td>
<td><code>wtbman -name wsbroker1 -host finance -port 9999 -user tom -trimagents 3</code></td>
</tr>
<tr>
<td>Stop a local Transaction Server</td>
<td><code>wtbman -name wsbroker1 -stop</code></td>
</tr>
<tr>
<td>Stop a remote Transaction Server</td>
<td><code>wtbman -name wsbroker1 -host finance -port 9999 -user tom -stop</code></td>
</tr>
</tbody>
</table>

^2 Prompts for a password.
Index

&DISPLAY 69
&OUT 69
&OUT-FMT 69
&OUT-LONG 69
&WEBSTREAM 69
<Sendrange>Debugging
  firewall configurations 110
<Sendrange>Firewalls
  debugging 110

A

Accessing
  IP Packet Filter settings 104
  Web server 108
  WebSpeed agent 109
  WebSpeed broker 109
AdminServer
  defined 22
  WebSpeed in Windows 35
Agent
  defined 22
  starting dynamically 47
  trimming running 47
Agent Variables WebTool 64
Apache Web server 33
AppBuilder
  Code Section Editor 55
  connecting to a database server 56
  defined 21
  development modes 57
  documentation 58
  main window 55
  object palette 54
  specifying a broker 56
  specifying a browser 56
  starting 55
  templates 54
  wizards 54
Application files
  appropriate directories 42
Architecture
  WebSpeed 24

B

Blank Template 54
broker
  defined 22
  setting up 88
browse.html 70

Browser (HTTP) response times 111

C

CGI Messenger
  executable 49
CGI Wrapper
  template 54
CGIIP executable name
  hiding 90
Changing
  script directory names 90
Code Section Editor 55
Compiling Web objects 43
Configurations
  Messenger-only installs 48
  overview 31
  restarting 73
Configuring
  ubroker.properties file for firewall 103
Control handlers
  defined 68

D

Data Browser WebTool 63
Data integrity
  restarting configurations 73
Database
  locking r-code 96
  servers 56
Databases WebTool 64
DBAUTHKEY 96
Debugging
  firewall configurations 104
Deployment model 95
Detail Wizard 54
Developer's Corner WebTool 65
Development modes 57
DMZ 99
Domain Name System 112

E

Editor WebTool 63
Editors 55
Environment variables
  PROPATH 41, 43
  WRKDIR 44
Examples
  directory 70
F
File Tools 63
Firewalls
  configuration 102
  configuring ubroker.properties for 103
  debugging 104
  with DMZ 102
Frameset Template 54
frameset.html 70

G
GENUUID utility 36

H
Help WebTool 65
hmapmain.i 70
HTML files
  deploying 42
HTML Mapping
  wizard 54
html-map.w 70
HTTP/S performance 111

I
IIS, See Microsoft IIS
include.i 70
inet_ns 102
Internet Information Services, See Microsoft IIS
IP Packet Filter settings
  accessing 104
ISAPI Messenger
  executable 49

J
Java class files
  deploying 42

L
Load balancing
  NameServer 39
Local development mode 57
Log files
  configuring 44
  maintaining 45

M
Main Template 54
main.html 70
Messages WebTool 64
Messenger
  debugging 108
  defined 23
  downloading executables 49
  executable location in Windows 49
  managing 52
  Messenger-only deployment 23
  operating system compatibility 49
  overview 48
  performance 111
  script file location 51
Method directory 70
Method procedures
  defined 68
Microsoft IIS
  Messenger executable 49
Multi-homed servers 112
Multiple IP address servers, See Multi-homed servers
Multiple Web servers 112

N
n-tier deployment
  Messenger-only installs 48
NameServer
  checking access using NSMAN -name NS1 -query 109
  defined 23
  linking with other NameServers 79
  load balancing 39
  "No NameServer" configuration 40
  predefined instance 38
  validating configurations 36
Netscape
  Messenger executable 49
Network traffic
  securing 88
"No Nameserver" configuration 40
Nonvisual objects 54
NSAPI Messenger
  executable 49
NSAPI-type Web server
  obj.conf file 50
NSMAN -name NS1 -query 108

O
obj.conf file
  modifying 50
  sample 50
Object palette 54
Object State WebTool 64
objects directory 70
Offsets
  deploying 43
### Index

**P**
- Parameters passing 98
- Passing unique identifiers 98
- Performance
  - browser response times 111
  - HTTP/S 111
  - multiple Web servers 112
  - WebSpeed Messengers 111
- priorityWeight property 39
- PRO*Tools defined 21
- procedur.p 70
- PROPATH
  - minimizing 98
  - PROPATH environment variable 41, 43
- ProPath WebTool 64
- Property file
  - overview 36

**R**
- Remote development mode 57
- Report Template 54
- Report Wizard 54

**S**
- Sample applications running 114
- Script directory names
  - changing 90
- script.html 70
- Scripting Lab WebTool 64
- Secure firewall configuration 102
- Security
  - application 88
  - changing script directory names 90
  - Firewalls 99
  - hiding CGIP executable name 90
  - locking r-code to the database 96
  - Microsoft IIS 91
  - modifying web-disp.p 96
  - network traffic 88
  - passing parameters 98
  - PROPATH 98
  - Web server 89
  - WebSpeed 87, 99
  - WebSpeed agent production setting 96
  - WebSpeed application 95, 98
  - WebSpeed Messenger Administration tool 92
  - WebSpeed server 93, 95

Source files 70
- Sports2000 database 114
- SportsPro Sales Advisor running 115
- Starting configurations
  - deployment network with separate Web server 82
  - development network with central machine 76
  - multiple LAN development environment 79
  - shared development/deployment network 84
  - single-machine 74
- Static files
  - location 32
- Super procedures 70
- Support directory 70

**T**
- Table template 54
- table.html 70
- tagmap.dat file
  - defined 68
  - deploying 43
- template directory 70
- Templates 54, 70
- Tools
  - PRO*Tools 21
  - WebTools 21
- Transaction broker
  - default directory
  - creating 41

**U**
- ubroker.properties file
  - configuring for firewall 103
  - editing 36
  - hierarchy 36
  - location 36
  - modifying for firewall 102
  - overview 36
  - unique values needed 36
  - validating edits 36
- WebSpeed Messenger logging 108
- Unified Broker framework described 38
- UNIX
  - Web server security 91
- URLs
  - WebSpeed broker 56
- uuid
  - in ubroker.properties 36
Index

V

classes of Web objects

Web objects

compiling 43
deploying 42
making secure 88

Web server

Apache 33
checking response 90
configuring 32
hiding type and version 90
Microsoft IIS 33
multiple 112
securing 89
testing 33
UNIX 91

WebSpeed

application security 95, 98
architecture 24
broker 56
changing script directory names 90
components 23
configuration in Windows 31
deployment architecture 21
examples directory 70
firewalls 99
hiding CGIIP executable name 90
method directory 70
Microsoft IIS 91
minimizing PROPATH 98
modifying web-disp.p 96
objects directory 70
passing parameters 98
securing network traffic 88
security 87, 99
source files 70
super procedures 70
support directory 70

WebSpeed (continued)

template directory 70
templates 54
UNIX Web servers 91
Web request flow chart 24
wizards 54

WebSpeed Agent

accessing 109
defined 22
production setting 96

WebSpeed architecture
requirements 24

WebSpeed broker

accessing 109
defined 22

WebSpeed components

overview 21

WebSpeed Messenger 48
See also Messenger

WebSpeed server

security 93, 95

WebSpeed Transaction Server

components 22, 45
forcing immediate shut down 47
managing 47
overview 48
sample instance 38
starting 45
stopping with WTBMAN 47

WebSpeed Workshop

components 21

WebTools

defined 21

Weight factors

arbitrary 39
fail-over 40
load balancing 39
percentage-based 39
setting to zero 40

Wizards

templates 70

Working application directory
creating 41

Working directory
environment variable 44
wrap-cgi.w 70

WSASP Messenger
executable 49

WSCONFIG utility

described 38

WSMAdmin page
URL 52

WTBMAN utility 123