## Contents

Preface ........................................................................ Preface–1

### Part I Introduction

1. Overview of Server and Services Administration ......................... 1–1
   AppServer for OpenEdge .............................................. 1–2
   AppServer Internet Adapter ........................................... 1–3
   OpenEdge Web Services Adapter .................................... 1–4
   WebSpeed Transaction Server and Messenger .......................... 1–5
   OpenEdge Adapter for SonicMQ ...................................... 1–6
   OpenEdge Adapter for Sonic ESB ..................................... 1–7

### Part II AppServer and Internet Adapter Administration

2. Configuring and Managing the AppServer ................................. 2–1
   Run-time components and operation .................................. 2–2
   Distribution of components ............................................ 2–5
   Fault-tolerant NameServers .......................................... 2–5
   Fault-tolerant AppServers and load balancing ....................... 2–6
   Operating modes ....................................................... 2–7
   IPv6 networks ......................................................... 2–8
   SSL-enabled AppServer operation .................................. 2–8
   AppServer startup and shutdown ..................................... 2–9
   AppServer administration framework ................................ 2–10
   AppServer clients ..................................................... 2–11
   OpenEdge Explorer, Progress Explorer, and management utilities . 2–12
   NameServers ......................................................... 2–13
   AppServer broker ..................................................... 2–14
   AppServer agents ..................................................... 2–14
   AppServer and NameServer properties file (ubroker.properties) .. 2–14
   Text editor and configuration utilities .................................. 2–14
   Preparing for AppServer administration ............................... 2–15
Configuring AppServer components .............................................. 2–16
  General steps for using OpenEdge Explorer or Progress Explorer to
  configure an AppServer instance ................................................ 2–16
  Configuring an AppServer with the Progress Explorer ..................... 2–20
  Editing the properties file .......................................................... 2–24
Setting up the environment for AppServer execution ....................... 2–27
  Environment variable settings ....................................................... 2–27
  Working directory settings .......................................................... 2–27
  AppServer and NameServer log files .............................................. 2–27
  OpenEdge remote SmartDataObject support .................................... 2–27
  Customizing the AppServer agent executable ................................ 2–28
Starting and managing an AppServer instance ............................. 2–29
  Requirements for starting an AppServer ........................................ 2–29
  Starting and managing an AppServer with OpenEdge Explorer
  or Progress Explorer ................................................................. 2–29
  Starting and managing an AppServer with the management utilities .... 2–31
Specifying the server pool parameters ....................................... 2–33
  Tuning for operating modes ........................................................ 2–33
  Managing AppServer agents ........................................................ 2–33
Managing code pages ............................................................... 2–34
  Code-page settings .................................................................... 2–34
  Run-time conversions ................................................................. 2–35
Checking for failed client connections ....................................... 2–36
  Failure detection and response ...................................................... 2–36
  Messages ................................................................................... 2–37
  Time out value tuning ................................................................. 2–37
  AppServer operating modes ........................................................ 2–37
  Configuring the AppServer .......................................................... 2–37
  Configuring clients .................................................................... 2–38
  HTTP, AIA, and SSL configurations .............................................. 2–41
Enabling Actional monitoring of AppServer resources .................. 2–42
  OpenEdge AppServer Interceptor ............................................... 2–42
  OpenEdge Adapter for Sonic ESB Interceptor ............................... 2–43
  OpenEdge Adapter for SonicMQ Interceptor .................................. 2–43
  WebSpeed Interceptor .................................................................. 2–44
  Web Services Adapter Interceptor ............................................... 2–44
  AppServer Internet Adapter (AIA) Interceptor ............................ 2–45
  OpenEdge Batch Client Interceptor ............................................. 2–45
  Web Services Out Interceptor ..................................................... 2–46
Summary of management tasks ................................................... 2–47

3. Configuring and Managing the AppServer Internet Adapter .......... 3–1
  Installing the AppServer Internet Adapter .................................... 3–2
    General AIA administration ......................................................... 3–2
    Security considerations for AIA administration ........................... 3–2
  Installing and configuring Web servers and Java servlet engines .... 3–4
    Enabling the Web server or JSE for SSL operation ....................... 3–6
  Configuring AIA components ....................................................... 3–7
  General steps for using OpenEdge Explorer or Progress Explorer
  to configure an AIA instance ......................................................... 3–7
  Configuring an AIA with OpenEdge Explorer ................................ 3–8
  Configuring an AIA with the Progress Explorer ............................ 3–9
  Configuring an AIA by editing the properties file ......................... 3–11
Viewing AppServer Internet Adapter connection and configuration information 3–13
  Configuration information ............................................................ 3–14
### Part III Web Services Adapter Administration

#### 4. Configuring a Web Services Adapter Installation
- Moving the WSA sample Web application
- Configuring the JSE to recognize the WSA
- Tuning the JSE to handle large Web service messages
- Modifying the WSA’s default Web service security settings
- Enabling the WSA for HTTPS client connections
- Installing WSA on a DMZ server

#### 5. Managing the Web Services Adapter
- WSA administration architecture
- WSA administration prerequisites
- Web server and JSE configurations
- Constructing URLs
- WSA as a JSE Web application
- Defining a Web application for the WSA
- Creating one or more WSA instances
  - Creating a WSA instance
  - Creating multiple WSA instances
- Starting and testing a WSA instance
  - Starting the WSA and its instances
  - Testing a WSA instance
- Configuring and managing a WSA instance
  - Properties of a WSA instance
  - Modifying properties of a WSA instance
  - Statistics of a WSA instance
  - Configuring the WSA for Web services

#### 6. Deploying and Managing OpenEdge Web Services
- Initializing a WSA instance’s default Web service properties
- Security considerations for Web service administration
- Deploying a Web service
  - Versioning Web Services
  - After deploying Web services
- Changing the Web service deployment directory
- Enabling a Web service
- Administering a deployed Web service
- Web service file management for deployed Web services
  - Options for providing the WSDL file to a client
  - Exporting and importing Web services
- Monitoring and tuning Web services
- Typical Web service administration scenarios
  - Deploying a new version of your Web service
  - Changing Web service properties

#### 7. Web Services Adapter Security Configurations
- Overview
- Role of the JSE
- Role of the WSA
- JSE and WSA security files
- Security features available
- Initial settings
Alphabetical list of configuration instructions .......................................................... 7–6
Controlling access to Web services, WSDL, and WSA administration
  using one JSE security-constraint .............................................................................. 7–7
Controlling access to Web services, WSDL, and WSA administration
  using user-authorization role-names ......................................................................... 7–8
Customizing the default administrator’s permissions and action
  settings ......................................................................................................................... 7–10
Customizing the PSCOper role’s permissions and action settings .................... 7–10
Customizing WSA administration roles ................................................................. 7–11
Disabling access to all Web services, to all WSDL, or to all
  WSA administration ................................................................................................. 7–12
Disabling authentication for WSA administration ................................................. 7–13
Disabling the deploying and undeploying of Web services ........................... 7–14
Disabling WSDL listings ............................................................................................ 7–14
Enabling authentication for Web services ............................................................... 7–15
Enabling authentication for WSDL ............................................................................. 7–17
Enabling multiple user roles to access Web services, WSDL, or
  WSA administration ................................................................................................. 7–18
Enabling Web services per user and per application ............................................ 7–20

8. Using the WSA Management Utility (WSAMAN). ......................................................... 8–1
  Overview of the WSAMAN utility ............................................................................. 8–2
  Syntax .......................................................................................................................... 8–2
  Functions for managing a WSA instance ................................................................. 8–2
  Functions for managing a Web service ...................................................................... 8–3

Part IV WebSpeed Administration

9. Configuring WebSpeed in Windows ............................................................................. 9–1
  WebSpeed configuration overview ............................................................................. 9–2
  WebSpeed administration ............................................................................................ 9–4
    The AdminService .................................................................................................... 9–4
    OpenEdge Explorer or Progress Explorer ............................................................. 9–5
    Configuring WebSpeed with the OpenEdge Explorer .......................................... 9–6
    Configuring WebSpeed with the Progress Explorer ............................................. 9–9
    WebSpeed command-line utilities ......................................................................... 9–12
    NameServer command-line utilities .................................................................... 9–13
  Setting up WebSpeed on the Web server machine ................................................. 9–14
    Installing the Messenger executable .................................................................. 9–14
    Configuring Netscape Web servers for NSAPI Messengers ............................... 9–15
    Restarting the Netscape Web server ..................................................................... 9–17
  Placing static files on the Web server ...................................................................... 9–18
  Overview of the ubroker.properties file ................................................................. 9–19
  Starting the AdminService ....................................................................................... 9–20
  Setting up the WebSpeed environment .................................................................... 9–21
    Setting environment variables .............................................................................. 9–21
  Configuring WebSpeed components ....................................................................... 9–23
    Eliminating the NameServer ................................................................................ 9–23
    Understanding the NameServer’s load-balancing option .................................... 9–25
  Starting the WebSpeed Transaction Server and NameServer ............................. 9–27
  Maintaining the WebSpeed Transaction Server and NameServer log files ..................... 9–29
  Configuring a WebSpeed Messenger-only installation ......................................... 9–30
  Starting WebSpeed to test the configuration ............................................................ 9–31
## 10. Configuring WebSpeed on UNIX

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebSpeed configuration overview</td>
<td>10–2</td>
</tr>
<tr>
<td>WebSpeed administration</td>
<td>10–4</td>
</tr>
<tr>
<td>The AdminServer</td>
<td>10–4</td>
</tr>
<tr>
<td>OpenEdge Explorer or Progress Explorer</td>
<td>10–4</td>
</tr>
<tr>
<td>WebSpeed command-line utilities</td>
<td>10–4</td>
</tr>
<tr>
<td>NameServer command-line utilities</td>
<td>10–5</td>
</tr>
<tr>
<td>Setting up WebSpeed on the Web server machine</td>
<td>10–6</td>
</tr>
<tr>
<td>Installing the Messenger executable</td>
<td>10–6</td>
</tr>
<tr>
<td>Configuring the Netscape Web server for the NSAPI Messenger</td>
<td>10–7</td>
</tr>
<tr>
<td>Restarting the Netscape Web server</td>
<td>10–9</td>
</tr>
<tr>
<td>Placing static files on the Web server</td>
<td>10–12</td>
</tr>
<tr>
<td>Overview of the ubroker.properties file</td>
<td>10–13</td>
</tr>
<tr>
<td>The ubroker.properties file structure</td>
<td>10–13</td>
</tr>
<tr>
<td>Editing the ubroker.properties file</td>
<td>10–13</td>
</tr>
<tr>
<td>Starting the AdminServer</td>
<td>10–16</td>
</tr>
<tr>
<td>Setting up the WebSpeed environment</td>
<td>10–17</td>
</tr>
<tr>
<td>Setting environment variables</td>
<td>10–17</td>
</tr>
<tr>
<td>Configuring WebSpeed components</td>
<td>10–19</td>
</tr>
<tr>
<td>Load balancing with the NameServer</td>
<td>10–19</td>
</tr>
<tr>
<td>Starting the WebSpeed Transaction Server and NameServer</td>
<td>10–20</td>
</tr>
<tr>
<td>Maintaining the WebSpeed Transaction Server and NameServer log files</td>
<td>10–22</td>
</tr>
<tr>
<td>Configuring a WebSpeed Messenger-only installation</td>
<td>10–23</td>
</tr>
<tr>
<td>Starting WebSpeed to test the configuration</td>
<td>10–24</td>
</tr>
<tr>
<td>Testing your configuration</td>
<td>10–25</td>
</tr>
<tr>
<td>Using the OpenEdge Explorer to check status</td>
<td>10–25</td>
</tr>
<tr>
<td>Using the query option to check status</td>
<td>10–25</td>
</tr>
<tr>
<td>Viewing the Messenger Administration page</td>
<td>10–26</td>
</tr>
<tr>
<td>Running the status.p procedure</td>
<td>10–26</td>
</tr>
<tr>
<td>Managing the WebSpeed Transaction Server</td>
<td>10–27</td>
</tr>
<tr>
<td>Dynamically starting additional agents</td>
<td>10–27</td>
</tr>
<tr>
<td>Trimming agents</td>
<td>10–27</td>
</tr>
<tr>
<td>Stopping the Transaction Server</td>
<td>10–28</td>
</tr>
<tr>
<td>Accessing help on WTBMAN</td>
<td>10–28</td>
</tr>
<tr>
<td>Managing the WebSpeed Messenger</td>
<td>10–29</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>11–2</td>
</tr>
<tr>
<td>Basic browser-agent interaction</td>
<td>11–3</td>
</tr>
<tr>
<td>How dynamic code-page support works</td>
<td>11–4</td>
</tr>
<tr>
<td>Summary of code-page conversions</td>
<td>11–6</td>
</tr>
<tr>
<td>Configuring the agent</td>
<td>11–7</td>
</tr>
</tbody>
</table>
Tasks for the WebSpeed developer ............................................ 11–8
  Determining the MIME code-page name .................................. 11–9
Storing the MIME code-page name ........................................... 11–10
  Accessing WEB-CONTEXT’s HTML-CHARSET attribute .............. 11–10
  Converting code-page names between MIME and OpenEdge formats . 11–11
Additional notes ......................................................................... 11–12

12. Connecting WebSpeed to a Data Source ................................ 12–1
  Connecting to an OpenEdge RDBMS ........................................ 12–2
    Connecting programmatically .............................................. 12–2
    Connecting to a local database ......................................... 12–3
    Connecting to a remote database ...................................... 12–3
  Connecting to a non-OpenEdge data source through a DataServer ... 12–5
    DataServer and WebSpeed architecture ................................. 12–6
    Setting up and running the DataServer ................................. 12–7
  Creating an international Web site ........................................ 12–8
    Setting up Transaction Servers .......................................... 12–9
    Configuring agents .......................................................... 12–9
    Directing Web requests .................................................... 12–11

13. WebSpeed Security ........................................................... 13–1
  Changing WebSpeed applications from development mode to production mode . 13–2
    Set the agent application mode to production ......................... 13–2
    Disable the WebSpeed Messenger Administration utility .......... 13–4
    Disabling WSMAdmin ....................................................... 13–5
    Establish the WebSpeed Messenger Administration Internet Protocol List (Optional) . 13–5
    Check the status of the Debug mode ................................... 13–6
  Changing additional settings to minimize security risks ............. 13–8
    Port numbers and WebSpeed server names ............................ 13–8
    Minimize the PROPATH entries ......................................... 13–8
    Techniques to minimize unauthorized access to WebSpeed Messengers ............................................. 13–9
    Restrict your file upload directory ..................................... 13–9
    WebSpeed configuration considerations ................................ 13–9
  Authenticating a password using SpeedScript ......................... 13–10
  Validating a password using SpeedScript ............................... 13–10
  Securing data transmissions between WebSpeed client and server components .................................. 13–12
    Secure HTTP Protocol (HTTPS) ........................................ 13–12
    Secure Sockets Layer (SSL) ............................................ 13–12
  Maximizing WebSpeed compatibility with your firewall ............. 13–15
    WebSpeed TCP port details .............................................. 13–15
    Important firewall considerations .................................... 13–16
    Network Address Translation (NAT) and the WebSpeed distributed configuration .... 13–16
    Using the NameServer client port range value settings with a firewall. 13–19

14. Using Active Server Pages with WebSpeed ............................. 14–1
  Setting up the WebSpeed ASP (WSASP) Messenger .................. 14–2
  Accessing the WebSpeed ASP Web page .................................. 14–3
  Enabling the WebSpeed ASP example .................................... 14–4
Part V Messaging and ESB Administration

15. OpenEdge Adapter for SonicMQ Administration .................. 15–1
   Introducing the OpenEdge Adapter for SonicMQ ................ 15–2
   Client connections to the OpenEdge Adapter for SonicMQ ..... 15–2
   Installing the OpenEdge Adapter for SonicMQ ................. 15–3
   SSL-enabled BrokerConnect operation ........................ 15–3
   HTTPS security for BrokerConnect .............................. 15–3
   Configuring and administering the OpenEdge Adapter for SonicMQ ........................................ 15–4
      Configuring ClientConnect ...................................... 15–4
      Configuring ServerConnect ...................................... 15–4
      Configuring BrokerConnect ..................................... 15–5
      Editing BrokerConnect attributes in ubroker.properties .................................................................. 15–7
      Configuring BrokerConnect from the command line .......................................................... 15–9
   Maximizing performance .............................................. 15–10
   Performance comparison .............................................. 15–10
   Optimizing message size ............................................. 15–10
   StreamMessage, MapMessage, and TextMessage ................ 15–10
   Remote and local calls .............................................. 15–10
   Message reuse ......................................................... 15–11
   Load balancing ......................................................... 15–11
   Discardable messages .............................................. 15–11
   Setting the CLASSPATH ............................................. 15–12
   Internationalization considerations ............................. 15–13

16. Configuring and Managing the OpenEdge Adapter for Sonic ESB ..... 16–1
   Introduction to the OpenEdge Adapter for Sonic ESB ........ 16–2
   Installation of the OpenEdge Adapter for Sonic ESB .......... 16–3
   Configuration .......................................................... 16–4
   Confirming the Sonic Management Console is configured properly (Optional) ................................... 16–5
   Using the OpenEdge Adapter for Sonic ESB ..................... 16–7
   Editing OpenEdge service properties ............................ 16–7
   Creating an OpenEdge service instance ......................... 16–10
   Editing an instance of an OpenEdge service .................... 16–15
   Exposing a service as a standard Web service .................. 16–16
   Deploying a service instance in Sonic ESB ....................... 16–18
   Security considerations for OpenEdge Adapter for Sonic ESB ............................................................. 16–20

Part VI Appendixes

A. Reference to OpenEdge Web Service Properties .................. A–1
   Overview .................................................................. A–2
   Summary of properties ............................................. A–2
   Setting Service properties ......................................... A–4
   Alphabetical reference ............................................. A–5
   appServiceHost ...................................................... A–5
   appServiceName ...................................................... A–5
   appServicePort ....................................................... A–5
   appServiceProtocol ................................................ A–5
   connectionLifetime ................................................ A–6
<table>
<thead>
<tr>
<th>Command and Utility Reference</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAPTCONFIG</td>
<td>B–3</td>
</tr>
<tr>
<td>ADAPTMAN</td>
<td>B–4</td>
</tr>
<tr>
<td>AIACONFIG</td>
<td>B–6</td>
</tr>
<tr>
<td>ASBMAN</td>
<td>B–7</td>
</tr>
<tr>
<td>ASCONFIG</td>
<td>B–12</td>
</tr>
<tr>
<td>ESBOEGEN</td>
<td>B–13</td>
</tr>
<tr>
<td>NSCONFIG</td>
<td>B–15</td>
</tr>
<tr>
<td>NSMAN</td>
<td>B–17</td>
</tr>
<tr>
<td>PROADSV</td>
<td>B–19</td>
</tr>
<tr>
<td>WSACONFIG</td>
<td>B–20</td>
</tr>
<tr>
<td>WSAMAN deploy</td>
<td>B–21</td>
</tr>
<tr>
<td>WSAMAN disable</td>
<td>B–23</td>
</tr>
<tr>
<td>WSAMAN enable</td>
<td>B–24</td>
</tr>
<tr>
<td>WSAMAN export</td>
<td>B–25</td>
</tr>
<tr>
<td>WSAMAN getdefaults</td>
<td>B–26</td>
</tr>
<tr>
<td>WSAMAN getprops (Service)</td>
<td>B–27</td>
</tr>
<tr>
<td>WSAMAN getprops (WSA)</td>
<td>B–28</td>
</tr>
<tr>
<td>WSAMAN getstats (Service)</td>
<td>B–29</td>
</tr>
<tr>
<td>WSAMAN getstats (WSA)</td>
<td>B–30</td>
</tr>
<tr>
<td>WSAMAN import</td>
<td>B–31</td>
</tr>
<tr>
<td>WSAMAN list</td>
<td>B–33</td>
</tr>
<tr>
<td>WSAMAN query (Service)</td>
<td>B–34</td>
</tr>
<tr>
<td>WSAMAN query (WSA)</td>
<td>B–35</td>
</tr>
<tr>
<td>WSAMAN resetdefaults</td>
<td>B–36</td>
</tr>
<tr>
<td>WSAMAN resetprops</td>
<td>B–37</td>
</tr>
<tr>
<td>WSAMAN resetstats (Service)</td>
<td>B–38</td>
</tr>
<tr>
<td>WSAMAN resetstats (WSA)</td>
<td>B–39</td>
</tr>
<tr>
<td>WSAMAN setdefault</td>
<td>B–40</td>
</tr>
<tr>
<td>WSAMAN setprops (Service)</td>
<td>B–41</td>
</tr>
<tr>
<td>WSAMAN setprops (WSA)</td>
<td>B–42</td>
</tr>
<tr>
<td>WSAMAN undeploy</td>
<td>B–43</td>
</tr>
<tr>
<td>WSAMAN update</td>
<td>B–44</td>
</tr>
<tr>
<td>WSCONFIG</td>
<td>B–45</td>
</tr>
<tr>
<td>WTBMAN</td>
<td>B–47</td>
</tr>
</tbody>
</table>
C. Reference to Dynamic Server Properties. ................................................. C–1
   Overview ................................................................................. C–2
   Dynamic properties .......................................................... C–3

Index ................................................................................. Index–1
Figures

Figure 2–1: AppServer run-time components ............................................. 2–2
Figure 2–2: AppServer administration framework ...................................... 2–10
Figure 3–1: AIA client connection information ......................................... 3–13
Figure 3–2: AIA configuration information ............................................... 3–14
Figure 5–1: WSA administration architecture ........................................... 5–3
Figure 5–2: Directory structure of the WSA as a JSE Web application ........... 5–6
Figure 12–1: The local DataServer ......................................................... 12–6
Figure 12–2: Remote DataServer .............................................................. 12–6
Figure 12–3: WebSpeed configuration that supports an international Web site ... 12–8
Figure 14–1: The WebSpeed ASP Web Page ........................................... 14–3
Figure 16–1: OpenEdge Native Services ............................................... 16–5
Figure 16–2: OpenEdge Web Services .................................................... 16–6
Tables

Table 2–1: AppServer run-time components .................................................. 2–3
Table 2–2: AppServer configuration entity names ..................................... 2–25
Table 2–3: AppServer agent status indications ........................................... 2–32
Table 2–4: OpenEdge AppServer Interceptor properties ............................. 2–42
Table 2–5: OpenEdge Adapter for Sonic ESB Interceptor properties .......... 2–43
Table 2–6: OpenEdge Adapter for SonicMQ Interceptor properties .......... 2–43
Table 2–7: Web Services Adapter Interceptor properties ........................... 2–44
Table 2–8: Web Services Adapter Interceptor properties ........................... 2–44
Table 2–9: AppServer Internet Adapter Interceptor properties ................... 2–45
Table 2–10: OpenEdge Batch Client Interceptor switches .......................... 2–45
Table 2–11: AppServer utilities ................................................................. 2–47
Table 2–12: Management tasks ................................................................. 2–48
Table 3–1: AIA configuration entity names ............................................... 3–11
Table 4–1: Moving the installed sample Web application ........................... 4–2
Table 5–1: URL components ....................................................................... 5–5
Table 5–2: Sample aliases for URL components ....................................... 5–5
Table 5–3: web.xml file items to check ...................................................... 5–8
Table 5–4: ubroker.properties properties of a WSA instance .................. 5–12
Table 5–5: Statistics of a WSA instance ...................................................... 5–14
Table 6–1: Setting the WSA instance’s default Web service properties .... 6–2
Table 6–2: Web service order info version 1 .............................................. 6–4
Table 6–3: Web service order info version 2 .............................................. 6–4
Table 6–4: File created when a Web service is deployed ......................... 6–5
Table 6–5: Actions for administering a deployed and enabled Web service 6–9
Table 6–6: Statistics for a Web service ...................................................... 6–11
Table 7–1: Initial settings for JSE security ................................................ 7–5
Table 7–2: Initial settings for WSA security .............................................. 7–5
Table 7–3: Controlling Web service, WSDL, and administration access using JSE security constraints ......................................................... 7–7
Table 7–4: Controlling Web service, WSDL, and administration access using role names ................................................................. 7–8
Table 7–5: Requiring Web service user authorization for Web service, WSDL, and administration access ................................................................. 7–9
Table 7–6: Disabling access to Web services, WSDL, and administration .... 7–13
Table 7–7: Setting the appAuth property ................................................... 7–16
Table 7–8: Setting the wsdlAuth property ................................................. 7–18
Table 7–9: Enabling multiple user roles for Web services, WSDL, and administration ................................................................. 7–18
Table 7–10: Setting security constraints for multiple user roles on Web services, WSDL, and administration ................................................................. 7–19
Table 7–11: Enabling Web services per user and per application .......... 7–19
Table 7–12: Sample Web service security constraints by application and role name ................................................................. 7–20
Table 7–13: Sample Web service user roles .............................................. 7–21
Table 8–1: WSA management functions .................................................... 8–2
Table 8–2: Web service management functions ...................................... 8–3
Table 9–1: Windows NT WebSpeed Messengers ...................................... 9–14
Table 9–2: Netscape Web server configuration ....................................... 9–15
Table 9–3: Weight factors based on percentage ...................................... 9–25
Table 9–4: Weight factors based on arbitrary sums .................................. 9–25
Table 10–1: UNIX WebSpeed Messengers .............................................. 10–6
Table 10–2: Netscape Web server configuration .................................. 10–7
Table 10–3: NSAPI Messenger settings .................................................. 10–8
Table 10–4: Options for invoking the CGI Messenger .......................... 10–10
Table 11–1: MIME and OpenEdge code-page equivalents .................. 11–9
Table 12–1: Remote connection parameters ........................................... 12–3
Table 13–1: Agent application mode descriptions ................................. 13–3
Table 13–2: Default port number to change .......................................... 13–8
Table 13–3: Network communications requirements ............................. 13–15
Table 13–4: Registration Mode connection—OpenEdge Explorer ............. 13–17
Table 13–5: RegistrationMode Connection—ubroker.properties file .......... 13–18
Table 15–1: AppServer configuration entity names .................................. 15–8
Table 15–2: BrokerConnect attributes .................................................. 15–8
Table A–1: Service properties .......................................................... A–2
Table A–2: requestWaitTimeout property values ..................................... A–9
Table A–3: serviceLoggingLevel property values .................................... A–10
Table B–1: ADAPTMAN command examples .......................................... B–5
Table B–2: Summary view output fields ............................................... B–8
Table B–3: Additional detail view output fields ....................................... B–9
Table B–4: ASBMAN command examples ............................................. B–11
Table B–5: NSCONFIG command examples ........................................... B–16
Table B–6: NSMAN command examples ............................................... B–18
Table B–7: PROADSV command examples ............................................ B–19
Table B–8: Setting the SOAP format for deployment using WSAMAN ......... B–22
Table B–9: Setting the SOAP format for import using WSAMAN ............... B–31
Table B–10: Setting the SOAP format for update using WSAMAN ............ B–44
Table B–11: WSCONFIG command examples ........................................ B–46
Table B–12: WTBMAN command examples .......................................... B–48
Table C–1: allowRuntimeUpdates property values ................................... C–2
## Procedures

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>web.xml</td>
<td>7–22</td>
</tr>
<tr>
<td>Using SMQConnect on a client example</td>
<td>15–4</td>
</tr>
</tbody>
</table>
Preface

This Preface contains the following sections:

- Purpose
- Audience
- Organization
- Using this manual
- Typographical conventions
- Examples of syntax descriptions
- OpenEdge messages
- Third party acknowledgements

Purpose

This manual provides a central point of reference for configuring and managing the following OpenEdge® Application Server products and OpenEdge adapters:

- OpenEdge AppServer™
- AppServer Internet Adapter
- OpenEdge Web Services Adapter
- WebSpeed® Transaction Server
- WebSpeed Messenger
• OpenEdge Adapter for SonicMQ®
• OpenEdge Adapter for Sonic ESB®

All of these products and adapters share one or more of the following common features:

• Comprise part of the OpenEdge Application Server product set
• Rely on the Unified Broker framework for configuration and administration
• Rely on the OpenEdge NameServer for fault-tolerance and availability
• Play a unique role within OpenEdge application and integration services to help you develop and deploy applications as part of a Service Oriented Architecture (SOA)

Audience

This manual is for you if you need to configure and manage any of the OpenEdge Application Server products and OpenEdge adapters that it features. This manual is also for you if you need to deploy AppServer applications, WebSpeed applications, ABL applications that include JMS messaging using SonicMQ, or if you need to deploy OpenEdge Web services within the OpenEdge or Sonic ESB environment. For more information on these products and adapters, see OpenEdge Getting Started: Application and Integration Services. For most of these products and components it is helpful to be familiar with the Unified Broker framework. For more information on this framework, see OpenEdge Getting Started: Installation and Configuration.

Organization

Part I, Introduction

Chapter 1, “Overview of Server and Services Administration”

Introduces the OpenEdge server and adapter products who’s management is described in this manual, and provides references for more information on product architecture and usage.

Part II, AppServer and Internet Adapter Administration

Chapter 2, “Configuring and Managing the AppServer”

Describes the components, tools, and procedures for managing an AppServer installation.

Chapter 3, “Configuring and Managing the AppServer Internet Adapter”

Describes the components, tools, and procedures for managing an AppServer Internet Adapter installation.

Part III, Web Services Adapter Administration
Chapter 4, “Configuring a Web Services Adapter Installation”

Describes the basic features of Web Services Adapter (WSA) management and some post
installation configuration tasks.

Chapter 5, “Managing the Web Services Adapter”

Describes how to create and manage a WSA Web application in the context of a Java
servlet engine (JSE), how to create and manage WSA servlets in the context of a WSA
Web application, and how to create and manage corresponding WSA instances in the
context of the OpenEdge environment using the Unified Broker framework.

Chapter 6, “Deploying and Managing OpenEdge Web Services”

Describes how to deploy and manage OpenEdge Web services in the context of a single
WSA instance, and how to export and import Web services between WSA instances.

Chapter 7, “Web Services Adapter Security Configurations”

Describes the components of WSA security and how to create and manage common WSA
security configurations, listed for reference by alphabetical order.

Chapter 8, “Using the WSA Management Utility (WSAMAN)”

Introduces the WSAMAN command-line utility to perform many of the WSA and Web
service management tasks that are otherwise performed using OpenEdge Explorer.

Part IV, WebSpeed Administration

Chapter 9, “Configuring WebSpeed in Windows”

Describes the basic requirements and procedures for configuring a WebSpeed installation
in Windows.

Chapter 10, “Configuring WebSpeed on UNIX”

Describes the basic requirements and procedures for configuring a WebSpeed installation
on UNIX.


Describes how to manage dynamic code-page support for internationalizing WebSpeed
installations.

Chapter 12, “Connecting WebSpeed to a Data Source”

Describes how to connect a WebSpeed application to an OpenEdge data source, such an
OpenEdge RDBMS or DataServer.

Chapter 13, “WebSpeed Security”

Describes the components of WebSpeed security and how to use them to manage secure
WebSpeed installations and applications.
Chapter 14, “Using Active Server Pages with WebSpeed”

Describes how to manage a WebSpeed installation to incorporate Microsoft Active Server Pages (ASP) in a WebSpeed application.

Part V, Messaging and ESB Administration

Chapter 15, “OpenEdge Adapter for SonicMQ Administration”

Describes the components, tools, and procedures for managing a SonicMQ Adapter installation to provide Java Message Service (JMS) messaging for an ABL client of a SonicMQ JMS.

Chapter 16, “Configuring and Managing the OpenEdge Adapter for Sonic ESB”

Describes the components, tools, and procedures for managing OpenEdge Web services as OpenEdge services installed and enabled through the Sonic ESB Adapter on the Sonic Enterprise Service Bus (Sonic ESB).

Part VI, Appendixes

Appendix A, “Reference to OpenEdge Web Service Properties.”

Describes how to access and provides an alphabetical reference to configuration and run-time properties used to manage both OpenEdge Web services in the OpenEdge environment and OpenEdge services in the Sonic ESB environment.

Appendix B, “Command and Utility Reference”

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.

Appendix C, “Reference to Dynamic Server Properties”

Lists the server properties that can be changed at runtime.

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, see the OpenEdge Product Documentation Overview page on PSDN: http://communities.progress.com/pcom/docs/DOC-16074.
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 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>Convention</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</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 simultaneous 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 sequential key sequence: you press and release the first key, then press another key. For example, <strong>ESCAPE H</strong>.</td>
</tr>
<tr>
<td><strong>Syntax:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Fixed width</strong></td>
<td>A fixed-width font is used in syntax statements, code examples, system output, and filenames.</td>
</tr>
<tr>
<td><strong>Fixed-width italics</strong></td>
<td>Fixed-width italics indicate variables in syntax statements.</td>
</tr>
<tr>
<td><strong>Fixed-width bold</strong></td>
<td>Fixed-width bold indicates variables with special emphasis.</td>
</tr>
<tr>
<td><strong>UPPERCASE fixed width</strong></td>
<td>Uppercase words are ABL keywords. Although these are always shown in uppercase, you can type them in either uppercase or lowercase in a procedure.</td>
</tr>
<tr>
<td>![three arrows]</td>
<td>This icon (three arrows) introduces a multi-step procedure.</td>
</tr>
<tr>
<td>![one arrow]</td>
<td>This icon (one arrow) introduces a single-step procedure.</td>
</tr>
<tr>
<td><strong>Period (.) or colon (:)</strong></td>
<td>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.</td>
</tr>
<tr>
<td>![large brackets]</td>
<td>Large brackets indicate the items within them are optional.</td>
</tr>
<tr>
<td>![small brackets]</td>
<td>Small brackets are part of ABL.</td>
</tr>
<tr>
<td>![large braces]</td>
<td>Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.</td>
</tr>
<tr>
<td>![small braces]</td>
<td>Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.</td>
</tr>
<tr>
<td>![vertical bar]</td>
<td>A vertical bar indicates a choice.</td>
</tr>
<tr>
<td>![ellipses]</td>
<td>Ellipses indicate repetition: you can choose one or more of the preceding items.</td>
</tr>
</tbody>
</table>
Examples of syntax descriptions

In this example, ACCUM is a keyword, and aggregate and expression are variables:

Syntax

\[
\text{ACCUM aggregate expression}
\]

FOR is one of the statements that can end with either a period or a colon, as in this example:

Syntax

\[
\text{FOR EACH Customer:}
\text{DISPLAY Name.}
\text{END.}
\]

In this example, STREAM stream, UNLESS-HIDDEN, and NO-ERROR are optional:

Syntax

\[
\text{DISPLAY [ STREAM stream ] [ UNLESS-HIDDEN ] [ NO-ERROR ]}
\]

In this example, the outer (small) brackets are part of the language, and the inner (large) brackets denote an optional item:

Syntax

\[
\text{INITIAL [ constant , constant ]}
\]

A called external procedure must use braces when referencing compile-time arguments passed by a calling procedure, as shown in this example:

Syntax

\[
\{ \&argument-name \}
\]

In this example, EACH, FIRST, and LAST are optional, but you can choose only one of them:

Syntax

\[
\text{PRESELECT [ EACH | FIRST | LAST ] record-phrase}
\]

In this example, you must include two expressions, and optionally you can include more. Multiple expressions are separated by commas:

Syntax

\[
\text{MAXIMUM ( expression , expression [ , expression ] ... )}
\]
In this example, you must specify \texttt{MESSAGE} and at least one \texttt{expression} or \texttt{SKIP [ (n) ]}, and any number of additional \texttt{expression} or \texttt{SKIP [ (n) ]} is allowed:

\textbf{Syntax}

\begin{verbatim}
MESSAGE { expression | SKIP [ (n) ] } ...
\end{verbatim}

In this example, you must specify \{ \texttt{include-file}, then optionally any number of \texttt{argument} or \texttt{&argument-name = "argument-value"}, and then terminate with \}:

\textbf{Syntax}

\begin{verbatim}
{ include-file
  [ argument | &argument-name = "argument-value" ] ... }
\end{verbatim}

\textbf{Long syntax descriptions split across lines}

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

In this example, \texttt{WITH} is followed by six optional items:

\textbf{Syntax}

\begin{verbatim}
WITH [ ACCUM max-length ] [ expression DOWN ]
  [ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]
  [ STREAM-IO ]
\end{verbatim}

\textbf{Complex syntax descriptions with both required and optional elements}

Some syntax descriptions are too complex to distinguish required and optional elements by bracketing only the optional elements. For such syntax, the descriptions include both braces (for required elements) and brackets (for optional elements).

In this example, \texttt{ASSIGN} requires either one or more \texttt{field} entries or one \texttt{record}. Options available with \texttt{field} or \texttt{record} are grouped with braces and brackets:

\textbf{Syntax}

\begin{verbatim}
ASSIGN { [ FRAME frame ] { field [ = expression ] } }
  [ WHEN expression ] } ... |
  { record [ EXCEPT field ... ] }
\end{verbatim}
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:

   ```
   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.

**Third party acknowledgements**

OpenEdge includes AdventNet - Agent Toolkit licensed from AdventNet, Inc. [http://www.adventnet.com](http://www.adventnet.com). All rights to such copyright material rest with AdventNet.

OpenEdge includes ANTLR (Another Tool for Language Recognition) software Copyright © 2003-2006, Terence Parr. All rights reserved. Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes software developed by the Apache Software Foundation ([http://www.apache.org/](http://www.apache.org/)). Copyright © 1999 The Apache Software Foundation. All rights reserved (Xerces C++ Parser (XML) and Xerces2 Java Parser (XML)); Copyright © 1999-2002 The Apache Software Foundation. All rights reserved (Xerces Parser (XML); and Copyright © 2000-2003 The Apache Software Foundation. All rights reserved (Ant). The names “Apache,” “Xerces,” “ANT,” and “Apache Software Foundation” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “Apache”, nor may “Apache” appear in their name, without prior written permission of the Apache Software Foundation. For written permission, please contact apache@apache.org. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes Concurrent Java software Copyright 1994-2000 Sun Microsystems, Inc. All Rights Reserved. -Neither the name of or trademarks of Sun may be used to endorse or promote products including or derived from the Java Software technology without specific prior written permission; and Redistributions of source or binary code must contain the above copyright notice, this notice and the following disclaimers: This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN MICROSYSTEMS, INC. OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN MICROSYSTEMS, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

OpenEdge includes DataDirect software Copyright © 1991-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for JDBC Type 4 driver); Copyright © 1993-2009 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for JDBC); Copyright © 1988-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for ODBC); and Copyright © 1988-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect64 for ODBC).

OpenEdge includes DataDirect Connect for ODBC and DataDirect Connect64 for ODBC software, which include ICU software 1.8 and later - Copyright © 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

OpenEdge includes DataDirect Connect for ODBC and DataDirect Connect64 for ODBC software, which include software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Copyright © 1998-2006 The OpenSSL Project. All rights reserved. And Copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved.

OpenEdge includes DataDirect products for the Microsoft SQL Server database which contain a licensed implementation of the Microsoft TDS Protocol.

OpenEdge includes software authored by David M. Gay. Copyright © 1991, 2000, 2001 by Lucent Technologies (dtoa.c); Copyright © 1991, 1996 by Lucent Technologies (g_fmt.c); and Copyright © 1991 by Lucent Technologies (rnd_prod.s). Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

OpenEdge includes software authored by David M. Gay. Copyright © 1998-2001 by Lucent Technologies All Rights Reserved (decssrtod.c; strtodg.c); Copyright © 1998, 2000 by Lucent Technologies All Rights Reserved (decssrtodf.c; strtord.c); Copyright © 1998 by Lucent
Preface

Technologies All Rights Reserved (dmisc.c; gdtoa.h; gethex.c; gmisc.c; sum.c); Copyright © 1998, 1999 by Lucent Technologies All Rights Reserved (gdtoa.c; misc.c; smisc.c; ulp.c); Copyright © 1998-2000 by Lucent Technologies All Rights Reserved (gdtoaimp.h); Copyright © 2000 by Lucent Technologies All Rights Reserved (hd_init.c). Full copies of these licenses can be found in the installation directory, in the c:/OpenEdge/licenses folder. Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

OpenEdge includes http package software developed by the World Wide Web Consortium. Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All rights reserved. This work is distributed under the W3C® Software License [http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231] in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

OpenEdge includes ICU software 1.8 and later - Copyright © 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.


OpenEdge includes Infragistics NetAdvantage for .NET v2009 Vol 2 Copyright © 1996-2009 Infragistics, Inc. All rights reserved.

OpenEdge includes JSTL software Copyright 1994-2006 Sun Microsystems, Inc. All Rights Reserved. Software distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes OpenSSL software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Copyright © 1998-2007 The OpenSSL Project. All rights reserved. This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com). Copyright © 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written
permission, please contact openssl-core@openssl.org. Products derived from this software may
not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written
permission of the OpenSSL Project. Software distributed on an "AS IS" basis, WITHOUT
WARRANTY OF ANY KIND, either express or implied. See the License for the specific
language governing rights and limitations under the License agreement that accompanies the
product.

OpenEdge includes Quartz Enterprise Job Scheduler software Copyright © 2001-2003 James
House. All rights reserved. Software distributed on an “AS IS” basis, WITHOUT
WARRANTY OF ANY KIND, either express or implied. See the License for the specific
language governing rights and limitations under the License agreement that accompanies the
product. This product uses and includes within its distribution, software developed by the
Apache Software Foundation (http://www.apache.org/).

OpenEdge includes code licensed from RSA Security, Inc. Some portions licensed from IBM
are available at http://oss.software.ibm.com/icu4j/.

OpenEdge includes the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright
©1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

OpenEdge includes Sonic software, which includes software developed by Apache Software
Foundation. All rights reserved. The names “Ant”, “Axis”, “Xalan,” “FOP,” “The Jakarta
Project”, “Tomcat”, “Xerces” and/or “Apache Software Foundation” must not be used to
endorse or promote products derived from the Product without prior written permission. Any
product derived from the Product may not be called “Apache”, nor may “Apache” appear in
their name, without prior written permission. For written permission, please contact
apache@apache.org.

OpenEdge includes Sonic software, which includes software Copyright © 1999 CERN -
European Organization for Nuclear Research. Permission to use, copy, modify, distribute and
sell this software and its documentation for any purpose is hereby granted without fee, provided
that the above copyright notice appear in all copies and that both that copyright notice and this
permission notice appear in supporting documentation. CERN makes no representations about
the suitability of this software for any purpose. It is provided “as is” without expressed or
implied warranty.

OpenEdge includes Sonic software, which includes software developed by ExoLab Project
(http://www.exolab.org/). Copyright © 2000 Intalio Inc. All rights reserved. The names
“Castor” and/or “ExoLab” must not be used to endorse or promote products derived from the
Products without prior written permission. For written permission, please contact
info@exolab.org. Exolab, Castor and Intalio are trademarks of Intalio Inc.

OpenEdge includes Sonic software, which includes software developed by IBM. Copyright ©
1995-2003 International Business Machines Corporation and others. All rights reserved.
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
associated documentation files (the "Software"), to deal in the Software without restriction,
including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or
sell copies of the Software, and to permit persons to whom the Software is furnished to do so,
provided that the above copyright notice(s) and this permission notice appear in all copies of the
Software and that both the above copyright notice(s) and this permission notice appear in
supporting documentation. Software distributed on an "AS IS" basis, WITHOUT
WARRANTY OF ANY KIND, either express or implied. See the License for the specific
language governing rights and limitations under the License agreement that accompanies the
product. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

OpenEdge includes Sonic software, which includes the JMX Technology from Sun Microsystems, Inc. Use and Distribution is subject to the Sun Community Source License available at http://sun.com/software/communitysource.

OpenEdge includes Sonic software, which includes software developed by the ModelObjects Group (http://www.modelobjects.com). Copyright © 2000-2001 ModelObjects Group. All rights reserved. The name “ModelObjects” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “ModelObjects”, nor may “ModelObjects” appear in their name, without prior written permission. For written permission, please contact djacobs@modelobjects.com.

OpenEdge includes Sonic software, which includes code licensed from Mort Bay Consulting Pty. Ltd. The Jetty Package is Copyright © 1998 Mort Bay Consulting Pty. Ltd. (Australia) and others.

OpenEdge includes Sonic software, which includes files that are subject to the Netscape Public License Version 1.1 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/NPL/. Software distributed under the License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is Mozilla Communicator client code, released March 31, 1998. The Initial Developer of the Original Code is Netscape Communications Corporation. Portions created by Netscape are Copyright 1998-1999 Netscape Communications Corporation. All Rights Reserved.

OpenEdge includes Sonic software, which includes software developed by the University Corporation for Advanced Internet Development http://www.ucaid.edu Internet2 Project. Copyright © 2002 University Corporation for Advanced Internet Development, Inc. All rights reserved. Neither the name of OpenSAML nor the names of its contributors, nor Internet2, nor the University Corporation for Advanced Internet Development, Inc., nor UCAID may be used to endorse or promote products derived from this software and products derived from this software may not be called OpenSAML, Internet2, UCAID, or the University Corporation for Advanced Internet Development, nor may OpenSAML appear in their name without prior written permission of the University Corporation for Advanced Internet Development. For written permission, please contact opensaml@opensaml.org.

OpenEdge includes the UnixWare platform of Perl Runtime authored by Kiem-Phong Vo and David Korn. Copyright © 1991, 1996 by AT&T Labs. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED “AS IS”, WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHORS NOR AT&T LABS MAKE ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

OpenEdge includes Vermont Views Terminal Handling Package software developed by Vermont Creative Software. Copyright © 1988-1991 by Vermont Creative Software.
OpenEdge includes XML Tools, which includes versions 8.9 of the Saxon XSLT and XQuery Processor from Saxonica Limited (http://www.saxonica.com/) which are available from SourceForge (http://sourceforge.net/projects/saxon/). The Original Code of Saxon comprises all those components which are not explicitly attributed to other parties. The Initial Developer of the Original Code is Michael Kay. Until February 2001 Michael Kay was an employee of International Computers Limited (now part of Fujitsu Limited), and original code developed during that time was released under this license by permission from International Computers Limited. From February 2001 until February 2004 Michael Kay was an employee of Software AG, and code developed during that time was released under this license by permission from Software AG, acting as a "Contributor". Subsequent code has been developed by Saxonica Limited, of which Michael Kay is a Director, again acting as a "Contributor". A small number of modules, or enhancements to modules, have been developed by other individuals (either written especially for Saxon, or incorporated into Saxon having initially been released as part of another open source product). Such contributions are acknowledged individually in comments attached to the relevant code modules. All Rights Reserved. The contents of the Saxon files are subject to the Mozilla Public License Version 1.0 (the "License"); you may not use these files except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/MPL/ and a copy of the license can also be found in the installation directory, in the c:/OpenEdge/licenses folder. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

OpenEdge includes Xs3P v1.1.3. The contents of this file are subject to the DSTC Public License (DPL) Version 1.1 (the "License"); you may not use this file except in compliance with the License. A copy of the license can be found in the installation directory, in the c:/OpenEdge/licenses folder. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is xs3p. The Initial Developer of the Original Code is DSTC. Portions created by DSTC are Copyright © 2001, 2002 DSTC Pty Ltd. All rights reserved.

OpenEdge includes YAJL software Copyright 2007, Lloyd Hilaiel. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. Neither the name of Lloyd Hilaiel nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Part I

Introduction

Chapter 1, Overview of Server and Services Administration
Overview of Server and Services Administration

Many of the server products that support application and integration services share common requirements and similar tools for server configuration and administration. This chapter describes all of these server products and where to find more information on using as well as managing them, as described in the following sections:

- AppServer for OpenEdge
- AppServer Internet Adapter
- OpenEdge Web Services Adapter
- WebSpeed Transaction Server and Messenger
- OpenEdge Adapter for SonicMQ
- OpenEdge Adapter for Sonic ESB

For a complete overview of application and integration services in OpenEdge®, introducing and describing how these server products work together, see OpenEdge Getting Started: Application and Integration Services. For information on installing these products, see OpenEdge Getting Started: Installation and Configuration.
AppServer for OpenEdge

The AppServer™ is the core of OpenEdge application and integration services and is the engine for running ABL business logic that can be made available to application clients as application services. Essentially, the AppServer is an ABL runtime client with no user interface, but instead provides a means for client applications to call its ABL procedures and user-defined functions remotely. Like most server products described in this manual, it relies on the Unified Broker framework for configuration and administration.

The AppServer is often used together with the OpenEdge NameServer to provide connection and server-level fault tolerance and facilitate application service availability. With the help of additional server products and adapters whose management is also described in this manual, the AppServer can make its application services available to all types of OpenEdge clients in many different configurations.

For information on developing application services and programming OpenEdge clients (especially ABL clients) using the AppServer, see *OpenEdge Application Server: Developing AppServer Applications*.

For more information on managing the AppServer, see Chapter 2, “Configuring and Managing the AppServer.”
AppServer Internet Adapter

The AppServer Internet Adapter (AIA) allows ABL clients and Open Clients (.NET and Java™) to access the AppServer or the OpenEdge Adapter for SonicMQ over the Internet. The AIA allows clients to access the AppServer via the open and standard communications mechanisms of HTTP and HTTPS, providing easy access to an AppServer on a wide-area network (WAN) and secure access through the features of the Secure Sockets Layer (SSL). Like the AppServer, it relies on the Unified Broker framework for configuration and administration.

The AIA is installed and runs as a Java servlet in most any Java Servlet Engine (JSE) or in the integrated JSE of a Web server, which provides the Internet access to the AppServer. As the interface between Internet clients and the AppServer, the AIA provides connection options that allow it to access an AppServer on behalf of the client directly or by accessing a NameServer configured to provide location-independent access to the AppServer.

For information on connecting to the AppServer of the Internet using the AIA or AIA/S, see OpenEdge Application Server: Developing AppServer Applications for ABL clients and OpenEdge Development: Open Client Introduction and Programming for Open Clients.

For more information on managing the AIA or AIA/S, see Chapter 3, “Configuring and Managing the AppServer Internet Adapter.”
OpenEdge Web Services Adapter

The Web Services Adapter (WSA) provides a communications and transformation gateway between industry Web service clients and AppServer application services. Using the WSA, you can deploy a Web service (OpenEdge Web service) developed from an existing AppServer application service and make it available to Web service clients using Web Service Description Language (WSDL). Like the AIA, the WSA relies on the Unified Broker framework for configuration and administration.

Also like the AIA, the WSA runs as a Java servlet in most any Java Servlet Engine (JSE) or in the integrated JSE of a Web server, which provides the Internet access that Web service clients require. However, in addition to the HTTP and HTTPS communications provided by the JSE/Web server, the WSA understands the Simple Object Access Protocol (SOAP) used to exchange service messages between Web service clients and the Web services that it manages. Thus, the WSA translates between service requests from Web service clients and application service responses from the AppServer, which it returns as Web service responses to the clients.

OpenEdge Web services rely on Open Client technology to develop the client interface required by Web service clients. For information on how to build OpenEdge Web services and access them from Web service clients, see OpenEdge Development: Web Services and OpenEdge Development: Open Client Introduction and Programming.

For more information on managing the WSA, including information on how to deploy and manage OpenEdge Web services, see Part III, “Web Services Adapter Administration.” The chapters in Part III cover all aspects of Web service deployment, run-time management, and security, as well as the management of the WSA itself.
WebSpeed Transaction Server and Messenger

The WebSpeed® Transaction Server and WebSpeed Messenger combine to allow an application written in SpeedScript® (the WebSpeed variant of 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 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.

For information on developing SpeedScript applications for WebSpeed deployment, see OpenEdge Getting Started: WebSpeed Essentials and OpenEdge Application Server: Developing WebSpeed Applications.

For information on managing the WebSpeed Transaction Server and WebSpeed Messenger, see Part IV, “WebSpeed Administration.” The chapters in Part IV cover all aspects of WebSpeed deployment, run-time management, and security, as well as the management of the WebSpeed Transaction Server and Messenger themselves.
OpenEdge Adapter for SonicMQ

The OpenEdge Adapter for SonicMQ® allows an ABL or SpeedScript application to become a Java Messaging Service (JMS) client, using the SonicMQ broker as the JMS backbone. With the OpenEdge Adapter for SonicMQ and SonicMQ, an application can engage in Point-to-Point or Publish-and-Subscribe messaging sessions with other JMS clients that also use SonicMQ. An ABL client application can also use the JMS extensions provided by SonicMQ to handle different message formats, such as XML, temp-tables, and ProDataSets. The client application accesses all SonicMQ functions provided by the adapter using the installed ABL-JMS API. The OpenEdge Adapter for SonicMQ itself relies on the Unified Broker framework or OpenEdge property files, specifically AdminServerPlugins.properties and JavaTools.properties, for configuration and administration depending on the connection option used.

For more information on the OpenEdge Adapter for SonicMQ architecture, see OpenEdge Getting Started: Application and Integration Services.

For information on developing applications that incorporate messaging via SonicMQ, see OpenEdge Development: Messaging and ESB.

For more information on managing the OpenEdge Adapter for SonicMQ, see Chapter 15, “OpenEdge Adapter for SonicMQ Administration.” For more information on SonicMQ and its installation, configuration, and management, see the SonicMQ product documentation.
OpenEdge Adapter for Sonic ESB

The OpenEdge Adapter for Sonic ESB® enables an OpenEdge service hosted on Sonic ESB to be accessed as part of workflow processes managed by Sonic ESB. Sonic ESB is an application integration framework that provides high performance, reliability, and security. Its service-based architecture supports the deployment of discrete applications (services) that exchange messages according to sophisticated automated workflow processes, as well as the exposure of deployed applications as industry Web services.

The OpenEdge Adapter for Sonic ESB supports two methodologies:

- **Native Invocation methodology** — Sonic ESB calls and AppServer application directly. Native Invocation relies on invocation files that are created in your OpenEdge development environment, and are integrated into the workflow process. Native invocation provides simplified exposure of ABL code and a simplified process for mapping ABL parameters to Sonic messages.

- **Web Service methodology** — Sonic ESB calls an AppServer application as a Web service. The OpenEdge Adapter for Sonic ESB converts SOAP messages to AppServer protocol on inbound client requests, and converts outbound AppServer protocol to SOAP messages. Sonic ESB uses Web Service Description Language (WSDL) to make the OpenEdge service available to Web service clients, much as does the WSA in the OpenEdge environment (see the “OpenEdge Web Services Adapter” section on page 1–4).

For detailed information on developing applications with both methodologies of the OpenEdge Adapter for Sonic ESB and working with OpenEdge services in Sonic ESB-integrated applications, see *OpenEdge Development: Messaging and ESB*.

Once installed in the Sonic ESB environment, the OpenEdge Adapter for Sonic ESB has no management interface of its own. Rather, you manage its deployed (installed) OpenEdge services on the Sonic ESB using the Sonic Management Console and related tools. For more information, see Chapter 16, “Configuring and Managing the OpenEdge Adapter for Sonic ESB.”
Part II

AppServer and Internet Adapter Administration

Chapter 2, Configuring and Managing the AppServer

Chapter 3, Configuring and Managing the AppServer Internet Adapter
This chapter describes the tasks required to configure, start up, shut down, and maintain the AppServer. The AppServer operates on UNIX and in Windows, and you can perform the required tasks from either the UNIX or Windows command lines or from OpenEdge Explorer or Progress Explorer.

AppServer administration includes common tasks for configuring all OpenEdge server products that use the NameServer (OpenEdge Unified Broker products). For an overview of these common tasks, including detailed information on NameServer configuration, see OpenEdge Getting Started: Installation and Configuration. The sections of this chapter describe:

- Run-time components and operation
- AppServer administration framework
- AppServer clients
- Text editor and configuration utilities
- Configuring AppServer components
- Setting up the environment for AppServer execution
- Starting and managing an AppServer instance
- Specifying the server pool parameters
- Managing code pages
- Summary of management tasks
Run-time components and operation

Figure 2–1 shows the major run-time components that compose the AppServer architecture and their basic relationships.

Figure 2–1: AppServer run-time components

The dotted arrows indicate optional communications to establish a connection between client applications and the AppServer. For more information on AppServer architecture, see OpenEdge Getting Started: Application and Integration Services.
Table 2–1 identifies and describes the components shown in Figure 2–1.

### Table 2–1: AppServer run-time components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client application</td>
<td>A process that requests the execution of remote ABL procedures in the context of an AppServer session. A client application can be:</td>
</tr>
<tr>
<td></td>
<td>• An ABL client session, including the WebClient</td>
</tr>
<tr>
<td></td>
<td>• A WebSpeed ABL (SpeedScript) session (WebSpeed agent), which runs ABL procedures on behalf of Web browser clients (for more information, see OpenEdge Application Server: Developing WebSpeed Applications)</td>
</tr>
<tr>
<td></td>
<td>• Another AppServer session—AppServer agent (for more information, see OpenEdge Application Server: Developing AppServer Applications)</td>
</tr>
<tr>
<td></td>
<td>• A .NET Open Client application (for more information, see OpenEdge Development: .NET Open Clients)</td>
</tr>
<tr>
<td></td>
<td>• A Java Open Client application (for more information, see OpenEdge Development: Java Open Clients)</td>
</tr>
<tr>
<td></td>
<td>• A client of OpenEdge Web services (including Sonic ESB services)</td>
</tr>
<tr>
<td></td>
<td>The OpenEdge interface in Figure 2–1 is the code that allows a client application to access an AppServer. For ABL clients, this interface is</td>
</tr>
<tr>
<td></td>
<td>accessed through built-in ABL statements and functions dedicated to AppServer access. For .NET, Java, and Web service Open Clients, this</td>
</tr>
<tr>
<td></td>
<td>interface is accessed through a client interface that is custom-built with the OpenEdge Open Client Toolkit to access AppServer procedures.</td>
</tr>
<tr>
<td>AppServer agent</td>
<td>A process that executes remote procedure requests in the context of an ABL session. Much like a batch ABL client, almost any ABL statement that you</td>
</tr>
<tr>
<td></td>
<td>can execute in an interactive ABL client you can execute within an AppServer agent. An AppServer instance typically contains multiple</td>
</tr>
<tr>
<td></td>
<td>AppServer agents that start up when you start the AppServer.</td>
</tr>
<tr>
<td></td>
<td>An AppServer agent can also act as an ABL client of another AppServer instance, by making its own remote procedure calls.</td>
</tr>
</tbody>
</table>
Configuring and Managing the AppServer

AppServer broker

A process that creates, manages, and allocates AppServer agents for access by client applications. The AppServer broker manages client connection requests and dispatches requests to AppServer agents. Exactly how it does this depends on the AppServer operating mode. (For more information, see *OpenEdge Application Server: Developing AppServer Applications.*) A single AppServer broker supports one AppServer instance.

NameServer

A process that directs client connection requests to an AppServer that supports a specified business function. A client indicates which AppServer instance it wants to connect to by specifying an application service name that identifies the required business function. (For more information on application services, see *OpenEdge Application Server: Developing AppServer Applications.*)

When an AppServer starts up that is configured for use with a NameServer, the AppServer registers the application services it supports with a specific NameServer. The NameServer with which an AppServer registers is the controlling NameServer for that AppServer instance.

For session-managed applications, when a client requests a connection to an application service, the NameServer directs the client to a single AppServer to handle all requests for the specified application service. For session-free applications, when a client requests a connection to an application service, the NameServer provides a list of available AppServer connections that OpenEdge uses to build a connection pool from which it locates an AppServer to handle each client request.

To provide fault-tolerant access to NameServers, you can configure multiple NameServer instances that handle the same application services. Each such NameServer is a replica of the others, thus providing connection-level fault-tolerance.

If you have installed the NameServer Load Balancer, multiple AppServers can also support the same application service, allowing the NameServer to distribute client connections (session-managed) or client requests (session-free) among them, thus providing server-level fault tolerance.

When using a NameServer to connect an AppServer (session-managed) or application service (session-free), the client only needs to know the host and port of the NameServer and the name of an application service. The ABL interface contacts the specified NameServer on behalf of the client, processes the response, and directs the client to the appropriate AppServer (session-managed) or AppServers (session-free).

---

Table 2–1: AppServer run-time components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppServer broker</td>
<td>A process that creates, manages, and allocates AppServer agents for access by client applications. The AppServer broker manages client connection requests and dispatches requests to AppServer agents. Exactly how it does this depends on the AppServer operating mode. (For more information, see <em>OpenEdge Application Server: Developing AppServer Applications.</em>) A single AppServer broker supports one AppServer instance.</td>
</tr>
<tr>
<td>NameServer(^1)</td>
<td>A process that directs client connection requests to an AppServer that supports a specified business function. A client indicates which AppServer instance it wants to connect to by specifying an application service name that identifies the required business function. (For more information on application services, see <em>OpenEdge Application Server: Developing AppServer Applications.</em>) When an AppServer starts up that is configured for use with a NameServer, the AppServer registers the application services it supports with a specific NameServer. The NameServer with which an AppServer registers is the controlling NameServer for that AppServer instance. For session-managed applications, when a client requests a connection to an application service, the NameServer directs the client to a single AppServer to handle all requests for the specified application service. For session-free applications, when a client requests a connection to an application service, the NameServer provides a list of available AppServer connections that OpenEdge uses to build a connection pool from which it locates an AppServer to handle each client request. To provide fault-tolerant access to NameServers, you can configure multiple NameServer instances that handle the same application services. Each such NameServer is a replica of the others, thus providing connection-level fault-tolerance. If you have installed the NameServer Load Balancer, multiple AppServers can also support the same application service, allowing the NameServer to distribute client connections (session-managed) or client requests (session-free) among them, thus providing server-level fault tolerance. When using a NameServer to connect an AppServer (session-managed) or application service (session-free), the client only needs to know the host and port of the NameServer and the name of an application service. The ABL interface contacts the specified NameServer on behalf of the client, processes the response, and directs the client to the appropriate AppServer (session-managed) or AppServers (session-free).</td>
</tr>
</tbody>
</table>

---

1. The same NameServer process can also coordinate Web browser access to WebSpeed Transaction Servers and ABL client access to OpenEdge DataServers or OpenEdge Adapter for SonicMQ. For more information on all NameServer features, see *OpenEdge Getting Started: Installation and Configuration.*

---
Distribution of components

The AppServer broker for an AppServer instance and its associated AppServer agents must all execute on the same computer. Client applications, however, can run on any computer in the network that can access the computer where the AppServer agent is running. Any required NameServer can execute on the same computer as a client application, the same computer as an AppServer instance, or on any other computer in the network to which both AppServer agents and client applications have access. With these conditions satisfied, OpenEdge can establish a connection between any client and any AppServer. For more information on how you can distribute AppServer components on a network, see the information on machine distribution in OpenEdge Getting Started: Installation and Configuration.

Fault-tolerant NameServers

You can configure NameServer such that a group of NameServers work together to resolve a client connection request. Having a group of NameServers working together to resolve the request provides fault-tolerant access to the NameServer function.

OpenEdge provides the following two mechanisms that you can use to implement fault-tolerant NameServers, and you can use them independently or together:

- **NameServer replication** — Where you configure multiple NameServer instances within a single subnet on different machines to listen on the same User Datagram Protocol (UDP) port. Clients send connection requests and AppServers send registration requests to all NameServer instances using UDP broadcasting. Broadcasting allows a set of replicated NameServers to be set up as one controlling NameServer for each AppServer instance.

- **NameServer neighbors** — Where you configure multiple NameServers on machines located in one or more subnets so that an initial NameServer instance receives the client connection request. If this initial NameServer cannot resolve the request, it passes the request on to a specified list of NameServer neighbors. These NameServer neighbors then attempt to resolve the connection request. Each NameServer neighbor represents the controlling NameServer for a separate AppServer instance.

For more information on setting up fault-tolerant NameServers, see OpenEdge Getting Started: Installation and Configuration.
Fault-tolerant AppServers and load balancing

When an AppServer registers with a NameServer, it specifies a list of application services. In the simplest configuration, only one AppServer instance can register a specific application service with the same controlling NameServer. Thus, the NameServer satisfies each request for the registered application service by directing the client connection to the same AppServer. However, if you install the NameServer Load Balancer, you can configure fault-tolerant AppServers by having multiple AppServer instances register the same application service with a single controlling NameServer.

Note: If you have not installed the NameServer Load Balancer and an AppServer tries to register the same application service on the same controlling NameServer of an already-registered AppServer, the AppServer attempting to register receives an error.

When a client requests an AppServer connection provided by fault-tolerant AppServers, the specified NameServer resolves the connection request using one of the registered AppServer instances that support the specified application service. By default, the NameServer chooses the AppServer instance at random from among the available instances that can satisfy the connection request, thus distributing connections equally.

The NameServer Load Balancer also allows you to have the NameServer distribute client connections across a set of fault-tolerant AppServers proportionately, according to a weight factor that you specify when you configure each AppServer instance. When each AppServer registers with the controlling NameServer, in addition to its location and supported application service list, the AppServer registers any weight factor you specify. The NameServer uses the weight factors configured for all instances in a set of fault-tolerant AppServers to determine how to distribute client connections among them.

The values of these weight factors are arbitrary. It is the relative differences between these values that determine how the NameServer balances load. To achieve effective load balancing, you must coordinate weight factor assignments according to individual AppServer performance. Thus, the most correct weight factor for each AppServer instance depends on the relative performance (for your application) of the particular AppServer platform and its configuration compared to other AppServers that register support for the same application service.

To perform these configuration tasks, you can use OpenEdge Explorer or Progress Explorer (in Windows only) or manually edit the ubroker.properties file.

Note that the NameServer is flexible enough that you can dynamically start an AppServer instance to register at any time among a set of running fault-tolerant AppServers. As an AppServer instance starts up and registers, the NameServer adds it to the list of available AppServers that support the same application services. The NameServer then apportions client connection requests according to the latest set of weight factors.

For more information on setting weight factors and the effects of these settings, see OpenEdge Getting Started: Installation and Configuration.
Operating modes

When you configure an AppServer instance, you must specify an operating mode for it. The operating mode determines how client requests are dispatched to individual AppServer agents running on the AppServer instance. Each operating mode features different performance and design trade-offs.

The AppServer supports the following four operating modes, in increasing order of complexity:

1. **State-reset** — All requests sent by a client connected to this AppServer go to the same AppServer agent. This AppServer agent remains dedicated to the same client for the life of the connection. When the client disconnects, the AppServer agent resets its context to what it was at startup, removing all context created during the terminated client connection.

   **Note:** This operating mode is conceptually similar to the operation of AppServers prior to Progress Version 9, which run in only one operating mode.

2. **State-aware** — All requests sent by a client connected to this AppServer go to the same AppServer agent. This AppServer agent remains dedicated to the same client for the life of the connection. When the client disconnects, the AppServer agent deletes any remote persistent procedures that are still active in its context. However, it maintains all other context created during the terminated client connection for access during future client connections. This context remains available until it is removed during a future client connection or the AppServer agent terminates.

3. **Stateless** — An AppServer agent is not dedicated to a specific client. The AppServer agent can execute a request from any client that has an outstanding request to the AppServer. Because a client request can execute in an AppServer agent that is different from any that executed previous requests for the same client, session context established for this client during a previous request might not be available. Because OpenEdge does not automatically clean up any session context between requests, the session context in which a client request runs might have been established by a request from a different client.

4. **State-free** — The AppServer is not dedicated to a specific client, but executes requests from all clients that connect to an application service that the AppServer supports. Its AppServer agents can execute requests from any client that is logically connected to and sends request to the application service that the AppServer supports. Because a client request can execute a request on any AppServer and agent that supports the application service, session context is usually not available from one request to the next for the same client.

For more information on the functional, performance, and design trade-offs among these operating modes, see *OpenEdge Application Server: Developing AppServer Applications*. 
IPv6 networks

You have the option of configuring any AppServer instance to communicate across an IPv6 network. Internet Protocol is the connectionless network layer protocol responsible for moving datagrams, or packets of information, from one node to another. Prior to Release 10.1C, OpenEdge network communications are implemented using IPv4. IPv4 continues to be the default protocol, and IPv6 must be explicitly requested at startup in your server properties file, or on the command line.

If no version is specified, then only IPv4 connections and communications are possible.

**Note:** Most IPv6 implementations allow an IPv4 connection to an IPv6 socket. This is called a V4 mapped address. Windows does not support V4 mapped addresses.

For information on using IPv6, see *OpenEdge Getting Started: Installation and Configuration*.

SSL-enabled AppServer operation

You have the option of configuring any AppServer instance to require Secure Sockets Layer (SSL) client connections. You can maintain both SSL-enabled and non-SSL AppServer instances, but a given instance supports only one type of connection, either secure or nonsecure.

Security derives from the client authentication of the server’s identity via a Public Key Infrastructure (PKI) and a symmetric data encryption system. To configure an AppServer instance for SSL operation, you must:

- Obtain and install a server private key and a public key certificate. OpenEdge provides built-in keys and certificates that are suitable for use on development or demonstration servers; for production machines, you should obtain server certificates from an internal or public Certificate Authority (CA).

- Specify an alias and password for access to the private key/digital certificate.

- Disable session caching, or enable it with a specified timeout.

- To perform these configuration tasks, you can use OpenEdge Explorer or Progress Explorer (in Windows only) or manually edit the `ubroker.properties` file.

To connect to an SSL-enabled AppServer, a client application must have access to a digital (public key) certificate (often called a CA Root Certificate) that can authenticate with the digital certificate used by the server, and the client must use a secure protocol.

For more information on SSL support in OpenEdge, see *OpenEdge Getting Started: Core Business Services*. 
**AppServer operating modes and SSL**

Supporting SSL tunneling imposes significant overhead on client/server transactions. To minimize the impact of this overhead, OpenEdge uses SSL tunneling between the client and either the AppServer agent or the AppServer broker, but not both, according to the operating mode in which the AppServer runs:

- **Session-managed** — For the session-managed modes (state-reset and state-aware), the client makes an SSL connection to the AppServer agent, after its initial connection to the broker. Because this initial client-broker connection does not carry customer-level information, SSL tunneling is not necessary.

- **Session-free** — For the session-free modes (stateless and state-free), the client makes an SSL connection to the AppServer broker, which is the single primary server connection. SSL tunneling is not necessary for the transmission of data between the broker and the AppServer agent, because this connection is local to a single system and therefore is not exposed to the network.

**AppServer startup and shutdown**

You typically configure AppServer instances and NameServers to start up at system startup time. The Unified Broker framework provides a graphical user interface and an equivalent set of command-line utilities that you can use to configure, start up, and manage NameServers and AppServer instances. For more information, see the “AppServer administration framework” section on page 2–10.

**SSL-enabled AppServer startup requirements**

An SSL-enabled AppServer instance cannot start if the correct key password is not provided, the server key store entry cannot be found, or the server digital certificate is out of date. See the “SSL-enabled AppServer operation” section on page 2–8 for more information.

**OpenEdge startup parameters**

You can also configure the AppServer agent to pass a set of OpenEdge startup parameters to any AppServer agents that it starts. These startup parameters have the same effect on each AppServer agent as the startup parameters used to start an ABL client. For example, one of the startup parameters you specify might be the Database (-db) parameter to indicate that all AppServer agents in the pool are to connect to a specific database when they start up.

**AppServer registration**

When you start an AppServer instance configured to use a NameServer, the first thing the AppServer agent does is to register with the controlling NameServer that you specified during AppServer configuration. When you shut down an AppServer instance, the AppServer agent immediately unregisters the AppServer with the controlling NameServer, ensuring that the NameServer no longer makes this AppServer available for client connection requests.
AppServer administration framework

The management functions for the AppServer are an integrated part of the Unified Broker framework. Figure 2–2 shows an overview of this framework as it applies to AppServer administration.

Figure 2–2: AppServer administration framework

Figure 2–2 shows the NameServer and AppServer components running on the same machine. You can also install the NameServer and AppServer on separate machines connected to the same network. In that case, a separate AdminServer and ubroker.properties file exist on each machine for access by OpenEdge Explorer.

For more information on distributed AppServer configurations, see OpenEdge Getting Started: Installation and Configuration.

The core of the Unified Broker framework is the AdminServer process, which resides on each machine where an OpenEdge server product is installed. (In Windows, this is the installed service, AdminService for OpenEdge.) The AdminServer performs the actual configuration and management of these products within the Unified Broker framework.
In addition to the AdminServer process, the AppServer administration framework consists of several supporting components (also see Figure 2–2):

- AppServer clients
- OpenEdge Explorer and management utilities
- NameServer
- AppServer agent
- AppServer agents
- The AppServer and NameServer properties file (ubroker.properties)
- Text editor and configuration validation utilities

For more information on how AppServer clients, NameServers, AppServer agents, and AppServer agents work together, see the sections on the AppServer in OpenEdge Getting Started: Application and Integration Services. Administration of these components follows the general procedures outlined in this section using OpenEdge Explorer, the ubroker.properties file, and the supporting utilities. For an overview of this configuration procedure and detailed information on configuring NameServers, see OpenEdge Getting Started: Installation and Configuration.

**AppServer clients**

This chapter describes configuration and startup of the AppServer itself. For information on configuring and starting up the various AppServer clients, see the documentation for your client product.

For ABL clients, see OpenEdge Getting Started: Installation and Configuration, and the following manuals:

- **Basic ABL client sessions** — OpenEdge Deployment: Managing ABL Applications
- **AppServer agents as client sessions of other AppServers** — This manual
- **WebSpeed agents** — This manual

For information on configuring and deploying .NET, Java, and Web services Open Clients, see OpenEdge Development: Open Client Introduction and Programming and OpenEdge Development: Web Services.
OpenEdge Explorer, Progress Explorer, and management utilities

OpenEdge provides different ways to manage local or remote AppServers. You can:

- Use the browser-based OpenEdge Explorer tool. OpenEdge Explorer is available with the installation of OpenEdge Management. It is also installed in any environment where Progress Explorer would be installed, or it can be installed separately. This tool provides access to the widest array of configuration settings.

- Use the older Progress Explorer tool (Windows only). Progress Explorer continues to be available in OpenEdge Release 10.2B. However, it does not provide access to new server features from the last few releases. Consider switching to OpenEdge Explorer whenever possible. Also note that Progress Explorer does not run on Windows 2008.

- Manually edit your properties file.

- Use the mergeprop utility installed with OpenEdge to manually edit the ubroker.properties file. For information on using mergeprop, see OpenEdge Getting Started: Installation and Configuration.

OpenEdge Explorer

OpenEdge Explorer is a web-based management tool available as a stand-alone installation or as part of OpenEdge Management. It offers access to the widest array of creation and configuration tasks.

There are different options to consider when installing OpenEdge Explorer. Install the tool by referring to OpenEdge Getting Started: Installation and Configuration. Once installed, you can find complete documentation in OpenEdge Management and OpenEdge Explorer: Getting Started or you can access the OpenEdge Explorer help file.

Progress Explorer

Progress Explorer is an older graphical user interface that provides an easy way for you to create and manage AppServers. Progress Explorer runs as a Windows client and works with the AdminServer (AdminService in Windows) in a client/server framework to manage AppServers and their controlling NameServers. This tool is not supported on Windows 2008 or later.

In general, Progress Explorer does not provide access to newer server features. Whenever possible, switch to using OpenEdge Explorer.

For an introduction to using Progress Explorer and information on creating and managing NameServers, see OpenEdge Getting Started: Installation and Configuration. For information on configuring an AppServer using Progress Explorer, see the “Configuring an AppServer with the Progress Explorer” section on page 2–20. For information on starting and managing an AppServer, see the “Starting and managing an AppServer with OpenEdge Explorer or Progress Explorer” section on page 2–29.
Command-line management utilities

The command-line management utilities run both in Windows and on UNIX and allow you to manage existing AppServer configurations. The management utilities for the AppServer include the:

- **NSMAN utility** — To manage NameServers
- **ASBMAN utility** — To manage AppServers

Like OpenEdge Explorer, the command-line management utilities work with the AdminServer in a client/server framework to manage AppServers and their controlling NameServers. Unlike OpenEdge Explorer, these utilities do not create a new AppServer configuration. Without OpenEdge Explorer, you must use a text editor and the configuration utilities. For more information, see the “Text editor and configuration utilities” section on page 2–14.

This chapter describes how to manage an AppServer using OpenEdge Explorer and the ASBMAN utility. For information on managing a NameServer using the NSMAN utility, see *OpenEdge Getting Started: Installation and Configuration*.

Using these utilities, you can locally or remotely:

- Start and stop an AppServer
- Manage AppServer agents
- Monitor the status of an AppServer

For information on starting and managing an AppServer using the management utilities, see the “Starting and managing an AppServer with the management utilities” section on page 2–31.

NameServers

The NameServer is an optional, but powerful, part of any AppServer configuration and management. Its features support the following AppServer capabilities:

- **Location transparency** for client connections
- **Load balancing** for multiple simultaneous client connections (optional)
- **Server-level fault tolerance** for client requests (optional)

For an overview of these features, see *OpenEdge Getting Started: Application and Integration Services*. For detailed information on configuring and managing NameServers to support all of these features, see *OpenEdge Getting Started: Installation and Configuration*. This chapter describes when and where to use NameServer instances as part of AppServer configuration and management.
AppServer broker

The AppServer broker performs the following functions:

- Registers the application services the AppServer provides with the controlling NameServer. For more information on application services, see OpenEdge Getting Started: Application and Integration Services.
- Manages connections between clients and a pool of AppServer agents that it starts.
- Maintains the status of each AppServer agent in its pool and scales the number of processes according to changing demand.
- When configured for stateless or state-free operating mode, dispatches remote requests to AppServer agents.

One AppServer agent provides connection management for a single AppServer instance. However, you can configure multiple AppServer instances, possibly using different operating modes and accessing different resources, for a single AppServer installation. For information on broker configuration options, see the “Configuring AppServer components” section on page 2–16.

AppServer agents

An AppServer agent is an OpenEdge session within the AppServer that executes ABL procedures on behalf of AppServer clients. The AppServer broker manages a pool of AppServer agents for an AppServer instance. Each AppServer agent in the pool runs an identical set of ABL procedures that share the same OpenEdge resources. The application services that an AppServer supports are really aliases for the same set of remote procedures supported by all AppServer agents in the pool.

When you configure AppServer agents, you specify a single set of properties that apply in common to all processes in the pool. For more information on configuring AppServer agents, see the “Configuring AppServer components” section on page 2–16.

AppServer and NameServer properties file (ubroker.properties)

The properties file (ubroker.properties) stores the component configurations of all OpenEdge Unified Broker products. For AppServer configurations, these components include the AppServer and the NameServer. There is one copy of this file residing in OpenEdge-Install-Directory/properties on each machine where OpenEdge is installed. So, if you install the NameServer on a separate machine, it has its own copy of the ubroker.properties file.

For general information on the ubroker.properties file and more specific information on AppServer properties, see the “Editing the properties file” section on page 2–24. For information on NameServer properties, see the sections on configuring the NameServer using the properties file in OpenEdge Getting Started: Installation and Configuration.

Text editor and configuration utilities

You can update the ubroker.properties file manually by either of the following methods:
AppServer administration framework

- Use the `mergeprop` utility installed with OpenEdge. For information on using `mergeprop`, see *OpenEdge Getting Started: Installation and Configuration*.

- Edit the file directly with any text editor.

To update NameServer and AppServer configurations from on UNIX, you must use one of the preceding methods. In general, you should update all UNIX configurations using OpenEdge Explorer. If you must update a UNIX AppServer configuration locally, make a copy of the file, update the copy, and then verify the result. OpenEdge provides the following configuration validation utilities that you can use to verify the correctness of any changes that you make to the file:

- **NSCONFIG utility** — To validate NameServer configurations. For more information on this utility, see *OpenEdge Getting Started: Installation and Configuration*.

- **ASCONFIG utility** — To validate AppServer configurations. For more information on this utility and on updating the `ubroker.properties` file manually, see the “Editing the properties file” section on page 2–24.

### Preparing for AppServer administration

You must complete the following preliminary tasks before you can begin the configuration and operation of an AppServer installation:

- Install the necessary product components. Typically, this involves installing OpenEdge and the AppServer on one or more network machines. If you plan to configure fault-tolerant servers or use load balancing, you must install the NameServer Load Balancer.

- Configure and set up the machines involved in the OpenEdge installation. Typically, you have completed any required network configuration for all machines before installing OpenEdge. After installation, you must also set up each machine environment to run OpenEdge and the AppServer.

- If you plan to use secure (SSL-enabled) AppServers, obtain and install a server private key and a public key certificate on each host machine. See the “SSL-enabled AppServer operation” section on page 2–8 for more information.

For more information on OpenEdge installation and setup, see *OpenEdge Getting Started: Installation and Configuration*. For more information on the distribution of resources in the AppServer environment, see the information on machine distribution for Unified Broker products in these same manuals. For more information on security-related concepts and configuring SSL-enabled OpenEdge servers, see *OpenEdge Getting Started: Core Business Services*. 
Configuring AppServer components

This section describes how to configure an AppServer. It first describes AppServer configuration with OpenEdge Explorer, then it describes how to configure an AppServer by editing the ubroker.properties file.

General steps for using OpenEdge Explorer or Progress Explorer to configure an AppServer instance

To access OpenEdge Explorer or Progress Explorer, open it from the OpenEdge program group. For more information, see OpenEdge Getting Started: Installation and Configuration. Once you have opened OpenEdge Explorer, you can configure an AppServer instance. The instructions below describe the general process. For specific instructions consult the help.

To configure AppServer instances:

1. Make sure that the AdminServer process is running on each of the following machines:
   - The machine where AppServers you want to configure are installed
   - The machine where all NameServers you want to configure are installed

   In Windows, when you complete the OpenEdge installation, the AdminServer automatically starts and is configured to auto-start whenever you reboot your system. You can also start the AdminServer (AdminService for OpenEdge) from the Windows Services applet available from the Control Panel. The AdminServer is running if you see both a java.exe and admserc.exe process listed in the Processes tab of the Task Manager.

   On UNIX, use the proadsv command to start the AdminServer. To check whether the AdminServer is running, run the ps command to show the full command line for each process on the system and locate any jre commands in the list. The AdminServer process is running if you see a jre command with the arguments that correspond to those specified for jvmstart in the OpenEdge proadsv shell script located in OpenEdge-Install-Directory/bin.

   For more information on using the PROADSV utility, see the “PROADSV” section on page B–19.

   For more information on starting the AdminServer, see OpenEdge Getting Started: Installation and Configuration.

2. In OpenEdge Explorer or Progress Explorer, connect to the running AdminServer processes that you verified in Step 1.
3. If you are using the NameServer, configure one or more NameServer instances to support all AppServer instances you want to configure. Note that for every AppServer instance that you plan to run on a separate machine from its controlling NameServer, you must configure a remote NameServer instance on the same machine as the AppServer. This remote NameServer instance must reference the host and port of the controlling NameServer for the AppServer. For more information, see the chapter on configuring OpenEdge Unified Broker products in *OpenEdge Getting Started: Installation and Configuration*.

4. Select where you want the AppServer instances to reside. If you are using the NameServer, configure the instances to register with the controlling NameServer (with or without load balancing). Specify any application service names required for clients to access the AppServer and any other required configuration information.

The sections that follow describe the basic steps for configuring each AppServer instance. For more information, see the OpenEdge Explorer or Progress Explorer online help.

**Configuring an AppServer with OpenEdge Explorer**

To configure the AppServer in Windows, you can use OpenEdge Explorer.

**To define and configure an AppServer instance:**

1. Make sure the AdminServer is running on the host where you want to configure the AppServer (see the “General steps for using OpenEdge Explorer or Progress Explorer to configure an AppServer instance” section on page 2–16).

2. Open OpenEdge Explorer and log in.

3. Connect to the AdminServer on your AppServer host (see *OpenEdge Getting Started: Installation and Configuration*).

4. To define a new AppServer or modify an existing AppServer, do one of the following:
   - To define a new AppServer, select the **AppServer** folder in OpenEdge Explorer’s tree view and click the **Add Resource** button. Choose **AppServer** and enter a unique name for the AppServer and click **Save**. Then open the property editor for the new instance by selecting the instance, right-clicking, and choosing **Properties**.
   - To modify an existing AppServer configuration, expand the **AppServer** folder in the tree view, select the AppServer instance you want to modify, right-click, and choose **Properties**.

The AppServer instance properties appear in a tabbed display to the left.
5. Select a property category tab and set the properties as required. You can accept the default values, if they are appropriate for your application. You probably want to specify the properties under each category. See the online help for detailed information about each property.

The Broker category specifies properties of the AppServer broker. Clicking this tab shows the following property subcategories:

- **General** — You must specify an Operating mode, and you probably want to specify a non-default value for the TCP/IP port number where the AppServer agent listens for requests.

  If you want the AppServer to start whenever you start the AdminServer, select the **Auto start** check box, and if you want to use a different working directory from the one specified during AppServer product installation, you can also change it here.

- **Owner Information** — You can optionally provide **Username** and **Password** information for the user who owns the AppServer instance.

- **Controlling NameServer** — You must indicate whether you plan to use a NameServer to control AppServer access. If so, check the **Register With NameServer** check box and select a controlling NameServer from the list of NameServer instances that you have already configured. Also specify how you want the NameServer to provide the host address of the AppServer broker for client connections.

- **AppService Name List** — You can either enter any names for the application services supported by this AppServer or select the **Supports default service** check box if you want this AppServer to support the default service for all client connections that do not specify an application service name. If you choose to use application service names, the default application service name is the name of the AppServer instance.

- **Logging Setting** — You can set the following logging options: specify a different pathname from the default for the broker log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous broker log file; specify a comma-separated list of logging entry types to be included in the broker log file, choosing from the valid values listed in the Progress Explorer online help; set a file-size threshold that determines the point at which a new log file is created (0 = unlimited log file size); and specify the maximum number of broker log files to be kept (0 = unlimited number of log files retained). See *OpenEdge Development: Debugging and Troubleshooting* for detailed information on logging options.

- **Advanced Features** — You can specify the maximum number of client connections (**Maximum client instances**) that the AppServer can support at one time, the AppServer weight factor (**Priority weight**) for load balancing, the time between retries to register the AppServer with the controlling NameServer, the timeout period for starting the AppServer, the timeout period for an AppServer to accept a client request, and the timeout period for the AppServer agent to trim its quota of AppServer agents between the maximum and minimum setting (see the **Agent** category). For more information on these options, see the Progress Explorer online help.
The **Agent** category specifies properties of the AppServer agents. Clicking this tab shows the following property subcategories:

- **General** — You can specify a pathname of the AppServer agent executable (**Server executable file**). You generally only need to specify a different value than the default if you build a new AppServer agent executable using the OEBuild utility. For more information, see the “Customizing the AppServer agent executable” section on page 2–28.

- Specify the OpenEdge startup parameters for the AppServer agent (**Server startup parameters**). These are the standard OpenEdge client startup parameters, and can include any parameters that you require for each AppServer session, including (but not limited to) all of the standard database, code-page, and process management parameters. For more information, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

- The AppServer can run with a different code page than the client application. For more information, see the “Managing code pages” section on page 2–34.

- For **PROPATH**, specify the semicolon-separated list of directories where the AppServer can locate ABL procedures to execute. This setting overrides any **PROPATH** environment variable settings on the AppServer host when it starts up. Make sure that all of the ABL procedures (r-code or source) that you want the AppServer to execute are located in one of these **PROPATH** directories. Otherwise, the procedure must be executed using its fully qualified pathname.

- Specify the minimum and maximum TCP/IP port numbers that the AppServer agent can assign to AppServer agents that it starts up. (Check with your system administrator for appropriate ranges.)

- **Logging Setting** — You can set the following logging options: specify a different pathname from the default for the server log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous server log file; specify a comma-separated list of logging entry types to be included in the server log file, choosing from the valid values listed in the online help; set a file-size threshold that determines the point at which a new log file is created (0 = unlimited log file size); and specify the maximum number of server log files to be kept (0 = unlimited number of log files retained). See *OpenEdge Development: Debugging and Troubleshooting* for detailed information on logging options.

- **Pool Range** — These settings determine the number of AppServer agents that the AppServer agent can start up and maintain for the AppServer. For more information on setting these values, see the “Specifying the server pool parameters” section on page 2–33.

- **Advanced Features** — To allow the ABL debugger to run in the AppServer session, select the **4GL debugger enabled** check box. Specify the names of any AppServer configuration procedures that you want the AppServer to execute, and any parameters for the Startup procedure. For more information on debugging AppServer applications and on AppServer configuration procedures, see *OpenEdge Application Server: Developing AppServer Applications*. 
The options in the SSL category options define the security settings for an SSL-enabled AppServer instance. Note that an AppServer enabled for SSL operation does not accept non-SSL client connections. For more information on SSL operations, see OpenEdge Getting Started: Core Business Services. Clicking this tab shows the following property subcategories:

- **General** — If you check the Enable SSL Client Connections box, select the alias for the private key/digital certificate entry (in the OpenEdge keystore) that you want to secure connections for this AppServer instance. Also, enter and confirm the password for this private key and digital certificate. You need not enter a password if you choose to use the default_server certificate and its default password.

- **Advanced Features** — By default, caching is enabled for the SSL client session, and you can enter a time-out value that specifies the length of time (in seconds) that a disconnected session is held in the cache. During this specified interval, a connected client can resume its session. To disable session caching, check the box.

The Messaging category specifies properties for an OpenEdge Adapter for SonicMQ ServerConnect (ServerConnect) process started by the application service running on this AppServer. It allows you to start a ServerConnect process at startup. You can also set logging options for the ServerConnect: specify a different pathname from the default for the broker log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous broker log file; specify a different pathname from the default for the server log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous server log file. See OpenEdge Development: Debugging and Troubleshooting for detailed information on logging options.

If you want to specify environment variables for AppServer execution, select the Environment Variables category. It allows you to enter name-value pairs for environment variable settings. Any values you set here override prior values set for the same environment variables in the operating system. For more information, see the “Environment variable settings” section on page 2–27.

**Note:** Do not set the PROPATH variable in the Environment Variables category. Use the Server General category instead.

6. Choose the OK button to save the configuration.

**Configuring an AppServer with the Progress Explorer**

To configure the AppServer in Windows, you can use Progress Explorer.

To define and configure an AppServer instance:

1. Make sure the AdminServer is running on the host where you want to configure the AppServer (see the “General steps for using OpenEdge Explorer or Progress Explorer to configure an AppServer instance” section on page 2–16).

2. Open Progress Explorer.
3. Connect to the AdminServer on your AppServer host (see *OpenEdge Getting Started: Installation and Configuration*).

4. To define a new AppServer or modify an existing AppServer, do one of the following:
   - To define a new AppServer, select the **AppServer** folder in the Progress Explorer’s tree view, right-click, and choose **New**. Enter a unique name for the AppServer and click **OK**. Then open the property editor for the new instance by selecting the instance, right-clicking, and choosing **Properties**.
   - To modify an existing AppServer configuration, expand the **AppServer** folder in the tree view, select the AppServer instance you want to modify, and open the AppServer property editor.

The AppServer instance property editor shows a tree view of property categories on the left and the properties for the selected category on the right.

5. Select a property category and click **Edit** to set the properties as required. You can accept the default values, if they are appropriate for your application. You probably want to specify the properties under each category. See the online help for detailed information about each property.

The **Broker** category specifies properties of the AppServer broker. Expanding this category shows the following property subcategories:

   - **General** — You must specify an Operating mode, and you probably want to specify a non default value for the TCP/IP port number where the AppServer agent listens for requests. If you want the AppServer to start whenever you start the AdminServer, select the **Auto start** check box, and if you want to use a different working directory from the one specified during AppServer product installation, you can also change it here.

   - **Owner Information** — You can optionally provide **Username** and **Password** information for the user who owns the AppServer instance.

   - **Controlling NameServer** — You must indicate whether you plan to use a NameServer to control AppServer access. If so, check the **Register With NameServer** check box and select a controlling NameServer from the list of NameServer instances that you have already configured. Also specify how you want the NameServer to provide the host address of the AppServer broker for client connections.

   - **AppService Name List** — You can either enter any names for the application services supported by this AppServer or select the **Supports default service** check box if you want this AppServer to support the default service for all client connections that do not specify an application service name. If you choose to use application service names, the default application service name is the name of the AppServer instance.
• **Logging Setting** — You can set the following logging options: specify a different pathname from the default for the broker log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous broker log file; specify a comma-separated list of logging entry types to be included in the broker log file, choosing from the valid values listed in the Progress Explorer online help; set a file-size threshold that determines the point at which a new log file is created (0 = unlimited log file size); and specify the maximum number of broker log files to be kept (0 = unlimited number of log files retained). See *OpenEdge Development: Debugging and Troubleshooting* for detailed information on logging options.

• **Advanced Features** — You can specify the maximum number of client connections (Maximum client instances) that the AppServer can support at one time, the AppServer weight factor (Priority weight) for load balancing, the time between retries to register the AppServer with the controlling NameServer, the timeout period for starting the AppServer, the timeout period for an AppServer to accept a client request, and the timeout period for the AppServer agent to trim its quota of AppServer agents between the maximum and minimum setting (see the Agent category). For more information on these options, see the Progress Explorer online help.

The **Agent** category specifies properties of the AppServer agents. Expanding this category shows the following property subcategories:

• **General** — You can specify a pathname of the AppServer agent executable (Server executable file). You generally only need to specify a different value than the default if you build a new AppServer agent executable using the OEBuild utility. For more information, see the “Customizing the AppServer agent executable” section on page 2–28.

• Specify the OpenEdge startup parameters for the AppServer agent (Server startup parameters). These are the standard OpenEdge client startup parameters, and can include any parameters that you require for each AppServer session, including (but not limited to) all of the standard database, code-page, and process management parameters. For more information, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

• The AppServer can run with a different code page than the client application. For more information, see the “Managing code pages” section on page 2–34.

• For **PROPATH**, specify the semicolon-separated list of directories where the AppServer can locate ABL procedures to execute. This setting overrides any **PROPATH** environment variable settings on the AppServer host when it starts up. Make sure that all of the ABL procedures (r-code or source) that you want the AppServer to execute are located in one of these **PROPATH** directories. Otherwise, the procedure must be executed using its fully qualified pathname.

• Specify the minimum and maximum TCP/IP port numbers that the AppServer agent can assign to AppServer agents that it starts up. (Check with your system administrator for appropriate ranges.)

• **Logging Setting** — You can set the following logging options: specify a different pathname from the default for the server log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous server log file; specify a comma-separated list of logging entry types to be included in the server log file, choosing from the valid values listed in the Progress Explorer online help; set a file-size
threshold that determines the point at which a new log file is created \((0 = \text{unlimited log file size})\); and specify the maximum number of server log files to be kept \((0 = \text{unlimited number of log files retained})\). See *OpenEdge Development: Debugging and Troubleshooting* for detailed information on logging options.

- **Pool Range** — These settings determine the number of AppServer agents that the AppServer agent can start up and maintain for the AppServer. For more information on setting these values, see the “Specifying the server pool parameters” section on page 2–33.

- **Advanced Features** — To allow the ABL debugger to run in the AppServer session, select the \textit{ABL debugger enabled} check box. Specify the names of any AppServer configuration procedures that you want the AppServer to execute, and any parameters for the Startup procedure. For more information on debugging AppServer applications and on AppServer configuration procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

The options in the SSL category options define the security settings for an SSL-enabled AppServer instance. Note that an AppServer enabled for SSL operation does not accept non-SSL client connections. For more information on SSL operations, see *OpenEdge Getting Started: Core Business Services*. Expanding this category shows the following property subcategories:

- **General** — If you check the \textit{Enable SSL Client Connections} box, select the alias for the private key/digital certificate entry (in the OpenEdge keystore) that you want to secure connections for this AppServer instance. Also, enter and confirm the password for this private key and digital certificate. You need not enter a password if you choose to use the \texttt{default_server} certificate and its default password.

- **Advanced Features** — By default, caching is enabled for the SSL client session, and you can enter a time-out value that specifies the length of time (in seconds) that a disconnected session is held in the cache. During this specified interval, a connected client can resume its session. To disable session caching, check the box.

The **Messaging** category specifies properties for an OpenEdge Adapter for SonicMQ ServerConnect (ServerConnect) process started by the application service running on this AppServer. It allows you to start a ServerConnect process at startup. You can also set logging options for the ServerConnect: specify a different pathname from the default for the broker log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous broker log file, specify a different pathname from the default for the server log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous server log file. See *OpenEdge Development: Debugging and Troubleshooting* for detailed information on logging options.
If you want to specify environment variables for AppServer execution, select the **Environment Variables** category. It allows you to enter name-value pairs for environment variable settings. Any values you set here override prior values set for the same environment variables in the operating system. For more information, see the “Environment variable settings” section on page 2–27.

**Note:** Do not set the `PROPATH` variable in the **Environment Variables** category. Use the **Server General** category instead.

6. Choose the **OK** button to save the configuration.

**Editing the properties file**

OpenEdge stores the configurations for both the NameServer and AppServer products in a properties file (`ubroker.properties`). The UNIX and Windows `ubroker.properties` files are the same except for platform-specific differences (for example, differences in directory path separators, and the differences between environment variable references on UNIX and registry references in Windows).

The properties file stores all the configuration definitions for all instances of the NameServer, and all instances of any AppServer, AppServer Internet Adapter, WebSpeed Server, Web Services Adapter, OpenEdge Adapter for SonicMQ, and DataServer product. Each configuration definition contains environment variables, and property settings for each product instance.

There is one copy of this file for each OpenEdge installation. Thus, if you install the NameServer on a separate machine from the AppServer product that it manages, the NameServer and AppServer product each have their own copy of the `ubroker.properties` file.

The AdminServer reads and updates this file according to your instructions using Progress Explorer and management utilities. The `ubroker.properties` file is installed in the properties subdirectory of the OpenEdge installation directory (for example, `OpenEdge-Install-Directory/properties/ubroker.properties` on UNIX, or `OpenEdge-Install-Directory\properties\ubroker.properties` in Windows).

**Guidelines for editing the properties file**

When editing the `ubroker.properties` file without Progress Explorer, note the following:

- You should not directly change the values in the `ubroker.properties` file unless you have a complete understanding of how the changes affect components. When possible, always use Progress Explorer to make all changes to this file.

**Note:** You can use the `mergeprop` utility installed with OpenEdge to manually edit the `ubroker.properties` file. For information on using `mergeprop`, see *OpenEdge Getting Started: Installation and Configuration*.

- You should always make a copy of this file, edit the copy, and verify the result before replacing the original with your edited copy.

- For complete definitions of all the properties and detailed information on how to set them, see the `ubroker.properties` README file, as well as the comments included in the properties file itself. Both files reside in the properties directory.
• Some properties are dynamic properties. A dynamic property is one that can be changed after the AppServer broker has been started. Depending on the property, changed values may be available to both existing and new agents or exclusively for new agents. Dynamic properties are only dynamic if the AppServer has been configured to support dynamic properties. See Appendix C, “Reference to Dynamic Server Properties” for a complete list.

The 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 products or product instances.

AppServer entries in the properties file

The AppServer configurations in `ubroker.properties` can include the entities listed in Table 2–2.

Table 2–2: AppServer configuration entity names

<table>
<thead>
<tr>
<th>Configuration entity name</th>
<th>Configuration entity function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[UBroker.AS.product-instance-name]</td>
<td>Defines property settings for this instance of an AppServer. The <code>ubroker.properties</code> file can contain several of these entities, each with a unique <code>product-instance-name</code>. For an AppServer, <code>product-instance-name</code> is the name you specify for the AppServer instance.</td>
</tr>
</tbody>
</table>

Thus, parent entities provide default values for all of their child entities. For example, the parent [UBroker] contains a set of definitions that can be inherited by its child AppServer product [UBroker.AS] and any other product entities, and then again by its child [UBroker.AS.product-instance-name] and any other product instance entities. However, at any child level, a redefinition of any value supersedes the default value of its parent. All children from the redefinition level down inherit this new value.
Editing and validating the properties file

To edit the properties file directly, use a text editor such as vi or Notepad. Once you edit the properties file, use the following utilities to validate the AppServer configuration information in the file:

- **NSCONFIG utility** — To validate NameServer configurations. For more information on this utility, see *OpenEdge Getting Started: Installation and Configuration*.

- **ASCONFIG utility** — To validate AppServer configurations. This section describes how to use this utility.

If the file contains any other OpenEdge server configurations (such as for WebSpeed or the OpenEdge Adapter for SonicMQ), run the configuration validation utilities for those Unified Broker products to ensure that these configurations are still valid. For more information, see *OpenEdge Getting Started: Installation and Configuration*.

**Note:** If you always use OpenEdge Explorer, you will never have to use these utilities.

The ASCONFIG utility displays the property settings associated with an AppServer configuration, and checks that the syntax and values are valid. You must run the ASCONFIG utility locally on the machine on which the AppServer is running. The utility does not run across the network.

This is the syntax used to invoke the ASCONFIG utility:

**Syntax**

```
asconfig [ [ [ -name AppServer-name ] [ -propfile path-to-properties-file ] [ -validate ] ] | -help ]
```

For more information on the ASCONFIG utility, see the “ASCONFIG” section on page B–12 and the “Summary of management tasks” section on page 2–47.

The following command validates the syntax and views the configurations of all AppServer instances defined within the `test.properties` file located in the current working directory:

```
asconfig -propfile test.properties -validate
```
Setting up the environment for AppServer execution

Before you start an AppServer instance you might need to perform the following tasks:

- Set any standard environment variables (for example, DLC) on the AppServer and NameServer machines
- Decide where to put your AppServer and NameServer log files
- Copy r-code files to support any OpenEdge SmartDataObjects you want to run on the AppServer
- Customize your AppServer agent executable using the OEBuild utility

Environment variable settings

Environment variables required by an AppServer are set during either AppServer installation or startup. You can modify these initial settings in the registry in Windows or in the shell scripts installed with OpenEdge on UNIX. You can also override most of these settings for an AppServer or NameServer using Progress Explorer. For more information, see the “Configuring AppServer components” section on page 2–16.

For more information on setting the standard OpenEdge environment variables, see the chapters on setting up the environment and configuring OpenEdge Unified Broker products in OpenEdge Getting Started: Installation and Configuration.

Working directory settings

The working directory settings for AppServer and NameServer each specify the directory in which the specified product starts up and, by default, where its log files and any other files output by AppServer applications are written. As with other settings, you can change or override the default settings for the working directory. You can also individually specify the locations of AppServer and NameServer log files. For more information, see the “Configuring AppServer components” section on page 2–16.

AppServer and NameServer log files

The AppServer and NameServer, together, generate three log files that you can use to monitor AppServer activity. For more information on working with AppServer and NameServer log files, see the chapter on configuring OpenEdge Unified Broker products in OpenEdge Getting Started: Installation and Configuration. For more information on the logging detail settings for AppServer agent log files, see the information on log file settings in OpenEdge Development: Debugging and Troubleshooting.

OpenEdge remote SmartDataObject support

To allow a remote SmartDataObject to run on the AppServer, copy the OpenEdge-compiled ADM (Application Development Methodology) r-code files to a directory in the AppServer PROPATH. You can find these files in the OpenEdge-Install-Directory\gui\adm2 directory, wherever you have the OpenEdge AppBuilder installed. For example, you can copy these files to your AppServer OpenEdge-Install-Directory\tty\adm2 directory. For more information on SmartDataObjects, see OpenEdge Development: AppBuilder.
Customizing the AppServer agent executable

The AppServer includes a standard AppServer agent executable that, in most cases, requires no customizing. However, you might have special needs that require you to add more software modules (configurable elements) to the executable. For example, the AppServer agent might be using a OpenEdge DataServer to connect to a non-OpenEdge database. In this case, you must include the appropriate DataServer when building the executable.

The AppServer agent executable is essentially a version of the ABL client. Like the ABL client, you can rebuild the AppServer agent executable using the OEBuild utility.

To build a new AppServer agent executable and make it available to your AppServer installation:

- Build the executable using the OEBuild utility according to the instructions provided in OpenEdge Deployment: Managing ABL Applications.
- Specify the pathname for your new AppServer agent executable using Progress Explorer or by setting the srvrExecFile property in the AppServer properties file (ubroker.properties). For more information on specifying the pathname for the AppServer agent executable, see the “Configuring AppServer components” section on page 2–16.

Note: The srvrExecFile property is a dynamic property. Any changes to the property affect all current and new brokers and agents.
Starting and managing an AppServer instance

OpenEdge provides two mechanisms to start and manage AppServers—Progress Explorer and the command-line management utilities. In general, you can start AppServer instances and the NameServer that controls them in any order. However, to allow clients to connect to any AppServer, you must first start the controlling NameServer for that AppServer.

You can also shut down a NameServer and the AppServer instances that it controls in any order. However, when you shut down the NameServer, any client applications that are not already connected cannot connect to the AppServer instances that this NameServer controls. Connected clients can still continue to operate with any running AppServers. If you restart the NameServer, any running AppServers that it controls automatically register with it and again become available for client connections.

Before you begin to develop or deploy applications in Windows or UNIX, note the requirements for each platform.

Requirements for starting an AppServer

Make sure that the AdminServer process is running on each of the following machines:

- The machine where the AppServer instance you want to start resides
- The machine where all required NameServers reside
- The machine where any OpenEdge databases or other DataServers reside that your AppServer instance needs to access

For more information, see the “General steps for using OpenEdge Explorer or Progress Explorer to configure an AppServer instance” section on page 2–16.

Starting and managing an AppServer with OpenEdge Explorer or Progress Explorer

You can used the browser-based OpenEdge Explorer to start a local or remote AppServer on any platform.

You can use Progress Explorer in Windows to:

- Start a local AppServer instance in Windows only
- Start a remote AppServer instance running in Windows or UNIX

To access OpenEdge Explorer or Progress Explorer, you can open them from the OpenEdge program group in Windows. For more options, see OpenEdge Getting Started: Installation and Configuration.

If you want to learn more about using OpenEdge Explorer, you can access the Help content in the tool. For a more thorough description of all the features related to OpenEdge Explorer, see OpenEdge Management and OpenEdge Explorer: Getting Started.

If you want to learn more about using Progress Explorer, the Help content in the tool is the best source.
To start an AppServer instance:

1. In either OpenEdge Explorer or Progress Explorer, connect to each of the running AdminServer processes required by your AppServer configuration. (See the Help of the tool you need for specific step-by-step information.) The name or IP address of the machine running each of these processes appears will be listed in the tool.

2. Navigate to the lists of NameServers and AppServers installed on the machine.

3. Start each NameServer that you need to support your AppServer instance and that is not already auto-started. For more information, see *OpenEdge Getting Started: Installation and Configuration*.

4. Start each OpenEdge database or DataServer that your application requires and that is not already auto-started. For more information on starting databases and DataServers, see *OpenEdge Getting Started: Installation and Configuration*.

5. Select the AppServer instance you want to start, and then start it.

You can also invoke the following management functions for the running AppServer instance:

- Stop the AppServer.

  **Note:** You cannot stop the AppServer if it is actively handling any client requests.

- Check and manage the operational status of the AppServer. You can also reduce and increase the number of running AppServer agents for this AppServer. For more information on the effects of these functions, see the “Specifying the server pool parameters” section on page 2–33. For more information on AppServer status indicators, see Table 2–3.

- View the log files for the AppServer.

- Delete the AppServer instance.

  **Note:** Before you can delete an AppServer instance, you must stop the AppServer.
Starting and managing an AppServer with the management utilities

You can use the command-line management utilities in Windows or on UNIX to start local AppServer instances and remote AppServer instances running on any platform.

To start an AppServer instance:

1. If your AppServer requires NameServer support, use the NSMAN utility to start each NameServer that you need to support your AppServer instance. For more information, see OpenEdge Getting Started: Installation and Configuration.

2. Start each OpenEdge database or DataServer that your application requires. For more information on starting databases and DataServers, see OpenEdge Getting Started: Installation and Configuration.

3. Use the ASBMAN utility to start the AppServer instance.

Using the ASBMAN utility

The ASBMAN utility runs on both in Windows and on UNIX. It allows you to invoke the following management functions on a local or remote AppServer instance:

- Start an AppServer.
- Check and manage the operational status of the AppServer. Management options allow you to reduce (trim) and increase (add) the number of running AppServer agents for this AppServer. For more information on the effects of these options, see the “Specifying the server pool parameters” section on page 2–33.
- Stop the AppServer.

Unlike Progress Explorer, the ASBMAN utility has no mechanism for viewing log files or deleting configured AppServer instances. If you want to set the AppServer log file or delete the AppServer instance, you must do it manually using operating system commands. To delete the AppServer, you must remove the entry for this AppServer instance in the AppServer properties file or use Progress Explorer. For more information on managing log files, see OpenEdge Getting Started: Installation and Configuration. For more information on accessing the AppServer properties file, see the “Editing the properties file” section on page 2–24.

Note: Before you can delete an AppServer instance, you must stop the AppServer.
This is the syntax to invoke the ASBMAN utility:

**Syntax**

```
ashman {
    { -name AppServer-name
      { -kill | -start | -stop | -query |
          -addservers number-to-start |
          -trimservers number-to-trim |
          -listclients |
          -clientdetail connection-handle
          -listallprops }
    }

    [ -host host-name -user user-name | -user user-name ]
    [ -port port-number ]
}
```

For more information on the ASBMAN utility, see the “ASBMAN” section on page B–7.

```
ashman -name AppServer-name -listclients -clientdetail connection-handle
```

You can check the status of a running AppServer using either OpenEdge Explorer or Progress Explorer or the -query option on the ASBMAN command. The returned status includes information on each AppServer agent running on the AppServer. The information on each process includes one of the status indications listed in Table 2–3.

### Table 2–3: AppServer agent status indications

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVAILABLE</td>
<td>The process is available to execute application logic.</td>
</tr>
<tr>
<td>BUSY</td>
<td>The process is actively executing application logic for an AppServer client. For a state-aware or state-reset AppServer, this status persists until the client disconnects from the AppServer.</td>
</tr>
<tr>
<td>LIMBO</td>
<td>The process is in a transitional state. If this status persists, it indicates an error condition.</td>
</tr>
<tr>
<td>LOCKED</td>
<td>On a stateless or state-free AppServer, the process is in a bound connection with an AppServer client. This status persists until the connection becomes unbound.</td>
</tr>
<tr>
<td>STARTING</td>
<td>The AppServer has started the process, but it has not yet completed initialization.</td>
</tr>
</tbody>
</table>
Specifying the server pool parameters

These parameters specify how many AppServer agents an AppServer starts and maintains to service clients. The number of running processes changes depending on these settings and how you are managing AppServer execution. While there is no exact set of rules for determining the correct settings for the server pool parameters, you can start with the following basic practical considerations:

- **Initial number of servers to start** — The number of processes started when the AppServer first starts up

- **Minimum servers** — The minimum number of processes that the AppServer keeps running to meet client demand

- **Maximum servers** — The maximum number of processes that the AppServer keeps running to meet client demand

Tuning for operating modes

In general, for a state-reset or state-aware AppServer, you probably want to set the Initial Server Instances to Start parameter to the average number of AppServer clients you expect so that each connecting client has efficient access to an AppServer agent. Thus, you are looking for the optimum settings for the number of concurrent connections. However, if your client load varies widely, you might set this value higher to offset the impact of client surges.

For a stateless and state-free AppServer, you must tune these parameters to handle changing client load based on how many clients that you can expect a single AppServer agent to handle in a given time period. Thus, you are looking for the optimum settings for the number of concurrent requests.

One way to assess this capacity for all operating modes is to set a high value for the Maximum Server Instances parameter and a low value for the Minimum Server Instances parameter. Then, run live for a period of time and see how many AppServer agents the AppServer tends to start. You might want to set the Initial Server Instances to Start parameter to that value.

Managing AppServer agents

In general, you can manage running AppServer agents based on the Server Pool Parameter settings as follows:

- You can trim any number of running AppServer agents below your Minimum Server Instances parameter value to zero.

- You cannot add any more AppServer agents than your Maximum Server Instances parameter setting. However, once you have found the maximum number of AppServer agents that you can run productively for the AppServer, you might even want to set that value lower if the setting negatively impacts other tasks on the system.
Managing code pages

The AppServer supports clients running with different code pages from the AppServer. When a client with a different code page invokes a remote procedure or user-defined function, the AppServer that receives the request converts character-string data between the client and AppServer code page as appropriate. The actual conversions are performed by the AppServer agent that responds to the request.

Code-page settings

You can set code pages for ABL clients and App Servers using several code page startup parameters. Each parameter allows you to specify any one of a wide variety of supported code pages for a specified data domain in the OpenEdge environment (for example, memory code page or input/output stream code page). For the AppServer, you specify code page startup parameters in OpenEdge Explorer or Progress Explorer as part of the $srvrStartupParam property setting in the ubroker.properties file. For more information on these parameters, see the chapter on code pages in OpenEdge Development: Internationalizing Applications. Whatever the settings, the AppServer performs the necessary conversions to and from the client application.

Notes: The $srvrStartupParam property is a dynamic property. Any changes to this property only affects new agents started after the value changes because the change affects how the agent is started. Existing (running) brokers and agents are not updated with the new value.

The Terminal Code Page (-cpterm) startup parameter has no meaning for AppServer.

Both .NET and Java Open Clients send requests (input) to the AppServer and receive responses (output) from the AppServer using Unicode UTF-8. The AppServer automatically converts between these Open Client Unicode UTF-8 requests and the particular code page that the AppServer is using.

Note: If your AppServer application provides database services to a multi-lingual open client application, using UTF-8 for both the AppServer and the database provides the most effective data interchange across application components.

Caution: Unicode supports text with embedded nulls. OpenEdge does not support this feature, no matter what character set it uses. Make sure that any open client applications filter character strings input to the AppServer for embedded nulls. Unfiltered input can result in unpredictable errors returned by the AppServer.
Run-time conversions

AppServer converts the following character data passed between a client and AppServer agent:

- Columns of input and output temporary tables
- Connection parameters
- Error messages
- Input and output parameters
- Return values (RETURN string) from ABL procedures
- Return values from ABL user-defined functions
- The names of executed ABL remote procedures or functions
Checking for failed client connections

When a client machine crashes or when a point in the network route between the server and a client fails, communication stalls and the AppServer may not be aware that a problem exists. Since the AppServer is a shared resource, failures of this type affect all application users. System resources like AppServer agents and memory are tied up and application resources like locked database records are unavailable to other users. It is important to be able to detect these failures and recover resources as quickly as possible.

The AppServer provides the ServerASK messaging protocol to help identify these failures. ASK stands for AppServer KeepAlive and ServerASK denotes that the direction of the keepalive messages is from the server to the clients. (The AppServer does not currently support a reciprocal ClientASK protocol.)

When an AppServer and client are configured to use ServerASK, the AppServer will periodically send a keepalive message to the client and wait for a response. If the client responds within the defined time, then the AppServer knows the connection is valid and resets the count down for the next keepalive message. If the client does not respond in the defined time, then the AppServer can presume there is a failed connection and respond appropriately, usually by disconnecting the socket.

Failure detection and response

The ServerASK protocol detects that a client bound to it is no longer connected as follows:

- The AppServer infers that the client connection has failed. It does this passively, by keeping track of the time of the last activity on the connection.

- If no messages have been sent to, or are received from, the client within a specific time interval (as specified by the serverASKActivityTimeout property), the AppServer will take active steps to determine if the client is still connected.

- The AppServer sends an askPing request to the client.

- When the client receives the askPing request, it must send an askPing response message to the AppServer, regardless of the current state of the connection.

- When the AppServer receives the askPing response, it updates the timestamp of the connection.

- If the AppServer does not receive a message from the client within a specific time interval (as specified by the serverASKResponseTimeout property), the AppServer will deem the connection to be in failure, and take action as if a fatal communication error had occurred. This will likely result in disconnecting the socket.

Note that the AppServer will not take action unless both the activity and response timers expire.
Checking for failed client connections

Messages

ServerASK only sends a message after an interval set by the serverASKActivityTimeout property expires and no activity has occurred between the client and AppServer. Each successful RunProcedure request and response rests the timer. If the AppServer issues a keepalive message and receives a keepalive response (or any other message from the client), then the connection timestamp is updated, the timer is reset, and the connection continues as normal. This minimizes the overhead required to use the protocol. You should set the property such that the interval is greater than the time it takes to process several normal operations from the client.

Note: Because an AppServer may be a newer or older software version than a client, the ServerASK protocol will establish the highest compatible version of the protocol to use for a particular connection.

Time out value tuning

Since the protocol relies on passive timers to determine failure and take action, there is always the risk that a busy client will be slow in sending a keepalive response message back to the AppServer. Careful tuning of the timer values is necessary to avoid this risk of disconnection of healthy client connections. Note that your AppServer implementation details may add several seconds to the time it actually takes for the AppServer to detect the timeout.

AppServer operating modes

The ASK protocol may be used regardless of the operating mode of the AppServer. When operating in state aware and state-reset modes, the ServerASK protocol support is provided by the AppServer agent process. However, in stateless and state-free modes, the AppServer broker performs the ServerASK duties, since the client process remains connected to the broker in these modes.

For the state-free operating mode, the ServerASK protocol is established independently on each connection in the connection pool.

Configuring the AppServer

To use this protocol, set the appServerKeepaliveCapabilities property in the [UBroker.AS] section of the ubroker.properties file. This property contains the ServerASK protocol capabilities supported by this AppServer. Valid values include:

- allowServerAsk
- denyServerAsk (default)

The serverASKActivityTimeout property determines the maximum amount of time (in seconds) from the last activity on a connection that the AppServer waits for a client message before initiating an askPing request. The minimum value is 30. The default value is 60.

The serverASKResponseTimeout property is the maximum amount of time (in seconds) that the AppServer will wait for an askPing response after an askPing request has been issued. The minimum value is 30. The default value is 60.
Configuring and Managing the AppServer

Note: All the properties described here can be set using OpenEdge Explorer or by manually editing the properties file.

Configuring clients

Clients that support ServerASK include:

- ABL client connections
- Java and .NET Open Client connections
- Web Services adapter connections
- Sonic Adapter connections
- SonicMQ Broker Adapter connections
- SonicESB Invocation Adapter connections
- SonicESB Web Services Adapter connections
- AppServer Internet Adapter connections

ABL client connections

To request the use of the ServerASK protocol on a connection, the ABL client must include the following new property in the connection string parameter of the `CONNECT()` method on the AppServer server handle:

`AppServerKeepalive capacitystring`

Indicates that the client would like to employ the AppServer Keepalive protocol on this connection, if supported and enabled by the AppServer. The absence of this property indicates that the (default) value for the ServerASK protocol will be used on this connection. The value `capacitystring` is either:

- `allowServerASK` (default)
- `denyServerASK`

The following is an example of an ABL `Connect()` method where the client requests the use of ServerASK:

```plaintext
```
Java and .NET Open Client connections

To request the use of ServerASK protocol on a connection, a Java or .NET Open Client must set the new connection property prior to instantiation of the appropriate AppObject. As with the existing connection properties, the new connection properties can be set on the static RunTimeProperties object (for properties that are applicable to all connections), or may be set on a Connection object instance for a specific connection. The values may be set or accessed using property names, or using the specific get and set methods for the property.

The PROGRESS.Session.AppServerKeepalive property is a string valued property that indicates the ASK feature that the client would like to employ on the connection.

The absence of this property indicates that the default value for the ServerASK protocol will be used on this connection. The value is either:

- allowServerASK (default)
- denyServerASK

**Note:** If you are connected synchronously to the AppServer using ServerASK and an Open Client, consider using the run-time property WaitIfBusy for .NET clients or setWaitIfBusy for Java clients to TRUE. Setting these properties to TRUE causes client requests to queue if the connection is busy. This protects you against the small possibility that the connection is busy with a ping from the ServerASK feature at the same time the client issues a request. In this case, the request fails because the connection is busy with the ServerASK request.

See OpenEdge Development: Java Open Clients and OpenEdge Development: .NET Open Clients for more information.

Web Services adapter connections

To request the use of ASK protocol for an application service, a Web Services adapter must set the connection property for that service:

**appServerKeepalive**

A string valued property that indicates the ASK features that the client would like to employ on the connection. The absence of this property indicates that the default value for the ServerASK protocol will be used on this connection. The value is either:

- allowServerASK (default)
- denyServerASK
Sonic Adapter connections

To request the use of the ServerASK protocol for an application service, a Sonic adapter must set new connection property for that service:

appServerKeepalive

A string valued property that indicates the ASK features that the client would like to employ on the connection. The absence of this property indicates that the default value for the ServerASK protocol will be used on this connection. The value is either:

- allowServerASK (default)
- denyServerASK

SonicMQ Broker Adapter connections

The ServerASK protocol supports for the SonicMQ Broker Adapter only. It is not supported for the ClientConnect or ServerConnect adapters. These adapters run on the same machine as the connecting client. As such, the ServerASK protocol does not provide any additional value for these adapters. The appServerKeepalive property is in the [Adapter] section of the ubroker.properties file.

SonicESB Invocation Adapter connections

The ServerASK protocol is supported for the SonicESB Invocation Adapter. The appServerKeepalive property is accessed using the Sonic Management Console.

To access these properties:

1. Start up the Sonic Management Console and select the Configure tab.
2. Expand the Services leaf and select OpenEdge Native Services.
3. Select the dev.OpenEdge service name.
4. Select the Runtime Properties button.

This will display a list of properties and their current values. The appServerKeepalive property is included in this list.

SonicESB Web Services Adapter connections

The ASK protocol is supported for the SonicESB Web Services Adapter. The appServerKeepalive property is accessed using the WSM editor after a service is defined using a WSM.
AppServer Internet Adapter connections

The new `appServerKeepalive` property defines the behavior of the AIA connections to the AppServer with regard to the ServerASK protocol. The new property is in the `[AIA]` section of the `ubroker.properties` file.

`appServerKeepalive`

Contains the ServerASK capabilities supported by the AppServer Internet Adapter. The value is either:

- `allowServerASK` (default)
- `denyServerASK`

HTTP, AIA, and SSL configurations

HTTP is a strict request/reply protocol in which only the client may initiate a request. Because, it is not possible for the AppServer to issue the askPing request to the client on this type of connection, the ServerASK protocol is not supported on connections between clients and AppServers using HTTP.

Because of this HTTP restriction, the ServerASK protocol may not be used on connections made through the AppServer Internet Adapter (AIA). However, because AIA acts as a proxy client for actual client applications, the ServerASK protocol can be used between AIA and the AppServer.

The ASK protocol does not support configurations that utilize Secure Sockets Layer (SSL) communications.
Enabling Actional monitoring of AppServer resources

Progress Actional is management software for widely distributed SOA applications. For an overview of Actional, see *OpenEdge Getting Started: Application and Integration Services*.

OpenEdge components that can be monitored by Actional have the necessary Interceptor functionality built in and disabled by default. If an OpenEdge system does not participate in Actional monitoring, the disabled Interceptor functionality has no effect on the performance of that system, since the functionality is disabled by default.

To turn an OpenEdge system into a managed node, you must first install and provision an Actional Analyzer on the system. You then enable the needed interceptors by configuring OpenEdge server properties. All OpenEdge server properties can be accessed by OpenEdge Explorer or edited directly in the `ubroker.properties` file. (You cannot access these properties from the older Progress Explorer tool.) The one exception to this is a Batch Client used to dispatch messages to an OpenEdge AppServer. For this case, you enable Actional through a startup parameter.

Actional has the notion of *group*, which is the top-level name for all components that share a group name in a managed node. For OpenEdge Interceptors, this is the application name as set by you. The `actionalGroup` server properties allow you to specify a group name.

Note that some of the Actional-related properties can be changed at runtime. These are referred to as dynamic properties.

The following sections describe the function of each OpenEdge interceptor and the properties used to configure each.

OpenEdge AppServer Interceptor

The OpenEdge AppServer Interceptor provides visibility for remote procedure calls into and out of the OpenEdge AppServer. The Interceptor supports all AppServer modes of operation.

The `ubroker` properties described in Table 2–4 support the Interceptor. These properties are also found on the *AppServer Agent Advanced Features* tab of OpenEdge Explorer.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Property type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>actionalEnabled</code></td>
<td>Boolean</td>
<td>False</td>
<td>This property enables and disables Actional monitoring. It supports dynamic updating, allowing the AppServer administrator to enable monitoring without restarting the AppServer.</td>
</tr>
<tr>
<td><code>actionalGroup</code></td>
<td>String</td>
<td>OpenEdge</td>
<td>This property sets the value used for Group when reporting to Actional.</td>
</tr>
</tbody>
</table>
OpenEdge Adapter for Sonic ESB Interceptor

Sonic ESB 7.5 and later include support for Actional monitoring. As a service running in a Sonic ESB container, the OpenEdge Adapter for Sonic ESB performs a ClientInteraction, retrieves the Actional manifest and passes it on to the AppServer it is calling. Enabling and disabling Actional monitoring is done at the ESB container level, and the adapter is enabled or disabled by reading the container-level settings, so that you do not have to explicitly enable this interceptor.

The `actionalGroup` property described in Table 2–5 and found in the Runtime Properties of the defined service sets the group name.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Property type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionalGroup</td>
<td>String</td>
<td>OpenEdge</td>
<td>This property sets the value used for the Group when reporting to Actional.</td>
</tr>
</tbody>
</table>

OpenEdge Adapter for SonicMQ Interceptor

The OpenEdge Adapter for SonicMQ Interceptor provides visibility for JMS messaging between an OpenEdge client and a SonicMQ broker. When the adapter is operating in BrokerConnect mode, it is treated as a server and performs its own client and server interactions. When operated in ClientConnect or ServerConnect mode, the adapter acts as a pass-through, forwarding any Actional manifest that an Actional-enabled OpenEdge client generates.

**Note:** While a Sonic 7.5 messaging broker will work in this scenario, a Sonic version 7.6 messaging broker is required to gain visibility into your messaging broker.

The ubroker properties described in Table 2–6 support the Interceptor. These properties are also found on the SonicMQ Adapter Broker Advanced Features tab of OpenEdge Explorer.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Property type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionalEnabled</td>
<td>Boolean</td>
<td>False</td>
<td>Specifying this property enables Actional monitoring for the application. It supports dynamic updating, allowing the MQ Adapter administrator to enable monitoring without restarting the Adapter.</td>
</tr>
<tr>
<td>actionalGroup</td>
<td>String</td>
<td>OpenEdge</td>
<td>This property sets the value used for the Group when reporting to Actional.</td>
</tr>
</tbody>
</table>
WebSpeed Interceptor

The WebSpeed Interceptor provides visibility for HTTP requests coming into a WebSpeed Transaction Server.

The ubroker properties described in Table 2–7 support the Interceptor. These properties are also found on the WebSpeed Agent Advanced Features tab of OpenEdge Explorer.

Table 2–7: Web Services Adapter Interceptor properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Property type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionalEnabled</td>
<td>Boolean</td>
<td>False</td>
<td>Specifying this property enables Actional monitoring for the application. This is a dynamic property.</td>
</tr>
<tr>
<td>actionalGroup</td>
<td>String</td>
<td>OpenEdge</td>
<td>This property sets the value used for the Group when reporting to Actional.</td>
</tr>
</tbody>
</table>

Web Services Adapter Interceptor

The OpenEdge Web Services Adapter Interceptor provides visibility for Web Service operations both into and out of the OpenEdge AppServer.

The ubroker properties described in Table 2–8 support the Interceptor. These properties are also found on the Web Services Adapter Advanced Features tab of OpenEdge Explorer.

Table 2–8: Web Services Adapter Interceptor properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Property type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionalEnabled</td>
<td>Boolean</td>
<td>False</td>
<td>Specifying this property enables Actional monitoring for the application.</td>
</tr>
<tr>
<td>actionalGroup</td>
<td>String</td>
<td>OpenEdge</td>
<td>This property sets the value used for the Group when reporting to Actional.</td>
</tr>
</tbody>
</table>
AppServer Internet Adapter (AIA) Interceptor

The AppServer Internet Adapter Interceptor provides visibility for remote procedure calls into and out of the OpenEdge AppServer when tunneled over HTTP.

The ubroker properties described in Table 2–9 support the Interceptor. These properties are also found on the AppServer Internet Adapter Advanced Features tab of OpenEdge Explorer.

Table 2–9: AppServer Internet Adapter Interceptor properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Property type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionalEnabled</td>
<td>Boolean</td>
<td>False</td>
<td>Specifying this property enables Actional monitoring for the application. It is not a dynamic property.</td>
</tr>
<tr>
<td>actionalGroup</td>
<td>String</td>
<td>OpenEdge</td>
<td>This property sets the value used for the Group when reporting to Actional.</td>
</tr>
</tbody>
</table>

OpenEdge Batch Client Interceptor

The OpenEdge Batch Client Interceptor provides visibility for a batch client receiving messages from an OpenEdge Adapter for SonicMQ, which then calls an AppServer to process the messages. This is useful for receiving JMS messages. When a message is received, the batch client performs a ServerInteraction. If the message consumer calls out to an AppServer, a ClientInteraction is performed, which is the same mechanism used for AppServer to AppServer messages. When the message consumer ends, the ServerInteraction ends.

The command line switches described in Table 2–10 support the Interceptor.

Table 2–10: OpenEdge Batch Client Interceptor switches

<table>
<thead>
<tr>
<th>Property name</th>
<th>Property type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionalEnabled</td>
<td>Boolean</td>
<td>False</td>
<td>Specifying this switch enables Actional monitoring for the application.</td>
</tr>
<tr>
<td>actionalGroup</td>
<td>String</td>
<td>OpenEdge:BatchClient</td>
<td>This switch specifies the group and service names, separated by a colon. If you do not specify this switch, the default value is used. If there is no colon in the value specified, it uses that value for the group name and the default value for the service name.</td>
</tr>
</tbody>
</table>
Web Services Out Interceptor

This interceptor allows an instrumented client (AppServer or Batch Client) to have visibility into the Web services calls they may make. The interceptor performs a ClientInteraction and adds an Actional manifest to the appropriate HTTP header for the Web services call. An instrumented Web service like Sonic ESB, Web services adapter, or an Actional-supported third party will be able to use the header and take part in the process flow.

There are no properties to enable or disable this interceptor. It relies on the properties set in the client where it is running (AppServer or Batch Client).
Summary of management tasks

This section summarizes the basic management tasks that are available to maintain existing AppServer configurations. They are described with reference to the command-line utilities, but you can also accomplish most of these tasks, except running the AdminServer, using OpenEdge Explorer or Progress Explorer.

Table 2–11 describes the utilities that OpenEdge provides to help configure and manage an AppServer installation, listed generally in order by task.

Table 2–11: AppServer utilities

<table>
<thead>
<tr>
<th>Utility</th>
<th>Execution location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASBMAN</td>
<td>Local or remote to the AppServer installation</td>
<td>Starts, stops, adds AppServer agents, trims AppServer agents, and queries status for an AppServer instance and its AppServer agent.</td>
</tr>
<tr>
<td>ASCONFIG</td>
<td>Local to the AppServer installation files</td>
<td>Validates the syntax of AppServer configurations in the ubroker.properties file. (Never needed if you always use OpenEdge Explorer or Progress Explorer.)</td>
</tr>
<tr>
<td>NSCONFIG</td>
<td>Local to the AppServer installation files</td>
<td>Validates the syntax of NameServer configurations in the ubroker.properties file. (Never needed if you always use OpenEdge Explorer or Progress Explorer.)</td>
</tr>
<tr>
<td>NSMAN</td>
<td>Local or remote to the NameServer machine</td>
<td>Starts, stops, and queries status for a NameServer.</td>
</tr>
<tr>
<td>PROADSV</td>
<td>Local to the AppServer installation</td>
<td>Starts, stops, and queries status for the AdminServer on UNIX. In Windows, you start the AdminService as a Windows service using the Services applet in the Control Panel.</td>
</tr>
</tbody>
</table>

For more information on the NameServer and NSMAN utility, and the AdminServer and PROADSV utility, see the “NSMAN” section on page B–17, the “PROADSV” section on page B–19, and OpenEdge Getting Started: Installation and Configuration.

Table 2–12 lists sample command-line utility syntax used to do the following:

- Start and stop the AdminServer in Windows
- Start, query, and stop the AdminServer on UNIX
- Verify the configuration of, start, query, and stop a NameServer instance
- Verify the configuration of, start, query, add AppServer agents to, trim AppServer agents from, and stop an AppServer instance
<table>
<thead>
<tr>
<th>To do this task . . .</th>
<th>Use these steps or commands . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start or stop the AdminService in Windows</td>
<td>From the Windows Control Panel, choose Administrative Tools, then choose Services; choose the AdminService for OpenEdge, and click either Start, Restart, or Stop</td>
</tr>
<tr>
<td>Start the AdminServer on UNIX</td>
<td>proadsv -start</td>
</tr>
<tr>
<td>Start the AdminServer on UNIX using a specified port</td>
<td>proadsv -port port-number -start</td>
</tr>
<tr>
<td>Query the AdminServer on UNIX</td>
<td>proadsv -host host-name -user user-name -query</td>
</tr>
<tr>
<td>Stop the AdminServer on UNIX</td>
<td>proadsv -stop -port port-number</td>
</tr>
<tr>
<td>View a NameServer configuration in the ubroker.properties file</td>
<td>nsconfig -name Name-Server</td>
</tr>
<tr>
<td>Validate the syntax and view the configurations of all NameServer instances defined within a specified property file</td>
<td>nsconfig -propfile property-file-path -validate</td>
</tr>
<tr>
<td>Start a local NameServer</td>
<td>nsman -name Name-Server -start</td>
</tr>
<tr>
<td>Start a remote NameServer</td>
<td>nsman -name Name-Server -host host-name -port port-number -user user-name -start</td>
</tr>
<tr>
<td>Query a local NameServer</td>
<td>nsman -name Name-Server -query</td>
</tr>
<tr>
<td>Query a remote NameServer</td>
<td>nsman -name Name-Server -host host-name -port port-number -user user-name -query</td>
</tr>
<tr>
<td>Stop a local NameServer</td>
<td>nsman -name Name-Server -stop</td>
</tr>
<tr>
<td>Stop a remote NameServer</td>
<td>nsman -name Name-Server -host host-name -port port-number -user user-name -stop</td>
</tr>
<tr>
<td>View an AppServer configuration in the ubroker.properties file</td>
<td>asconfig -name AppServer-name</td>
</tr>
<tr>
<td>Validate the syntax and view the configurations of all AppServer instances defined within a specified property file</td>
<td>asconfig -propfile property-file-path -validate</td>
</tr>
<tr>
<td>Start a local AppServer</td>
<td>asbman -name AppServer-name -start</td>
</tr>
<tr>
<td>Start a remote AppServer</td>
<td>asbman -name AppServer-name -host host-name -port port-number -user user-name -start</td>
</tr>
</tbody>
</table>
### Table 2–12: Management tasks

<table>
<thead>
<tr>
<th>To do this task . . .</th>
<th>Use these steps or commands . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query a local AppServer</td>
<td><code>asbman -name AppServer-name -query</code></td>
</tr>
<tr>
<td>Query a remote AppServer&lt;sup&gt;1&lt;/sup&gt;</td>
<td><code>asbman -name AppServer-name -host host-name -user user-name -query</code></td>
</tr>
<tr>
<td>Query summary of connected clients</td>
<td><code>asbman -name AppServer-name -listclients</code></td>
</tr>
<tr>
<td>Query detail of connected client</td>
<td><code>asbman -name AppServer-name -clientdetail connection-handle</code></td>
</tr>
<tr>
<td>Add AppServer agents to a local AppServer</td>
<td><code>asbman -name AppServer-name -addservers #</code></td>
</tr>
<tr>
<td>Add AppServer agents to a remote AppServer&lt;sup&gt;1&lt;/sup&gt;</td>
<td><code>asbman -name AppServer-name -host host-name -user -addservers #</code></td>
</tr>
<tr>
<td>Trim AppServer agents from a local AppServer</td>
<td><code>asbman -name AppServer-name -trimservers #</code></td>
</tr>
<tr>
<td>Trim AppServer agents from a remote AppServer&lt;sup&gt;1&lt;/sup&gt;</td>
<td><code>asbman -name AppServer-name -host host-name -user user-name -trimservers #</code></td>
</tr>
<tr>
<td>Stop a local AppServer</td>
<td><code>asbman -name AppServer-name -stop</code></td>
</tr>
<tr>
<td>Stop a local AppServer</td>
<td><code>asbman -name AppServer-name -host host-name -port port-number -user user-name -stop</code></td>
</tr>
</tbody>
</table>

1. Prompts for a password.

2. This is a NameServer defined on a remote machine, not a remote NameServer instance defined in the `ubroker.properties` file of the local machine.
The AppServer Internet Adapter (AIA) Web-enables the AppServer and the OpenEdge Adapter for SonicMQ BrokerConnect (BrokerConnect) by supporting HTTP and HTTPS protocols for sending information across the Internet. In addition, the HTTP and HTTPS protocols provide a way for clients to access an AppServer or a BrokerConnect connection when the client and server are separated by firewalls that limit connections to HTTP and HTTPS. These topics are explained in the following sections:

- Installing the AppServer Internet Adapter
- Installing and configuring Web servers and Java servlet engines
- Configuring AIA components
- Viewing AppServer Internet Adapter connection and configuration information

For more information on the architecture of the AIA and how it supports OpenEdge application services, see *OpenEdge Getting Started: Application and Integration Services*. For more information on OpenEdge Adapter for SonicMQ architecture, see *OpenEdge Getting Started: Application and Integration Services*. For information on how to build OpenEdge application services for the Internet and connect to them from ABL clients, see *OpenEdge Application Server: Developing AppServer Applications*. For information on how to connect Open Client applications to OpenEdge application services over the Internet, see *OpenEdge Development: Open Client Introduction and Programming*. You can also use the Progress Knowledge Center on the Web (http://www.progress.com/support) for information about Web-enabling ABL application clients of OpenEdge application services.
Installing the AppServer Internet Adapter

You can install the AppServer Internet Adapter (AIA) when you install the AppServer, or you can install AIA separately on another machine. You must have one of the following installed on the machine where you intend to install the AIA:

- A Java Servlet Engine (JSE), with HTTPS enabled if you are using Secure Sockets Layer (SSL) connections
- A Web server with an integrated JSE, with HTTPS enabled if you are using SSL connections

For more information on AIA installation, see OpenEdge Getting Started: Installation and Configuration.

General AIA administration

For information on general AIA administration and configuration, see the “Configuring AIA components” section on page 3–7. For information about installing and configuring Web servers, see the “Installing and configuring Web servers and Java servlet engines” section on page 3–4.

Security considerations for AIA administration

The AIA acts as an intermediary between the AppServer or BrokerConnect and clients that must access the server over the Internet. Thus, an application session involves two distinct connections, each of which is configured separately with respect to security.

The first connection is Internet-based between the AIA and the client. For this connection to be secure, the following conditions must be met:

- The client must use HTTPS protocol to send requests.
- The AIA must be HTTPS-enabled; that is, it must be configured to accept HTTPS requests from clients (via the JSE or Web server). To configure the AIA to accept HTTPS connection requests, you set the property httpsEnabled=1 by manually editing the ubroker.properties file. Or you can use OpenEdge Explorer or Progress Explorer.
- The JSE or Web server must support server authentication. Supporting server authentication requires that X.509 digital certificates be installed on both the Web server (or JSE) and the client machine. At each Web server to be accessed, a server certificate that uniquely identifies this Web server must be installed. As part of the SSL protocol, this server certificate is sent from the Web server to the client. See the “Enabling the Web server or JSE for SSL operation” section on page 3–6 for more information.
The second connection is via AppServer protocol between the AIA and the AppServer or BrokerConnect. For this connection to be secure, the following conditions must be met:

- The AIA must be **SSL-enabled**, meaning that it sends SSL data to the AppServer or BrokerConnect that is to process the client requests. To configure the AIA to send SSL requests, you set the property `sslEnable=1`. You set this property by manually editing the `ubroker.properties` file or by using OpenEdge Explorer or Progress Explorer. In addition, you must obtain and install public key certificates for the AIA host machine.

- The AppServer or BrokerConnect must be SSL-enabled, meaning that it accepts SSL requests from the AIA (or other clients). You set the property `sslEnable=1` by manually editing the `ubroker.properties` file or by using OpenEdge Explorer or Progress Explorer. You must also obtain and install a server private key and public key certificate and set additional SSL server properties.

**AIA behavior in an SSL environment**

A given AIA instance handles only one type of client request, either HTTP or HTTPS. The following results occur if the AIA receives a request via the incorrect protocol:

- If an HTTPS-enabled AIA instance receives an HTTP request, it redirects the request to HTTPS. That is, it returns the message to the client along with the appropriate URL to which the request should be resubmitted. This URL specifies HTTPS protocol and identifies the secure port on which the AIA listens for HTTPS requests.

- If an AIA instance that is not HTTPS-enabled receives an HTTPS request, it returns an `HTTPSNotEnabledException` error.

**Obtaining more information on SSL operations**

To obtain additional information about SSL operations, refer to the following sources:

- For more information on SSL support in OpenEdge, see *OpenEdge Getting Started: Core Business Services*.

- For more information on setting properties for the AIA and other Unified Broker products, see the `OpenEdge-Install-Directory\properties\ubroker.properties.README` file.

- For more information on managing digital certificates for ABL clients, see *OpenEdge Deployment: Managing ABL Applications*.

- For more information on managing digital certificates for Open Clients, see *OpenEdge Development: Open Client Introduction and Programming*. 
Installing and configuring Web servers and Java servlet engines

OpenEdge is tested with the latest releases of Tomcat for the Java servlet products (WSA and AIA). Tomcat is the reference JSE for testing. A different JSE can be used at your discretion but it must meet the JSE 2.1 standard.

A JSE must be installed for each Web server that you are using. How the JSE is installed and configured to run AIA depends on the JSE and the Web server. The information you provide during the installation and configuration varies depending on the Web server and JSE. However, some of the tasks you must perform and the information you must provide when you install and configure a JSE for the AIA are as follows:

- Add `OpenEdge-Install-Directory/java/aia.zip` to the JSE CLASSPATH.
- On UNIX only, add `OpenEdge-Install-Directory/lib` to the JSE LIBRARYPATH.
- Create a servlet instance for each AIA entry in the `ubroker.properties` file, and use `com.progress.aia.Aia` for the ClassName property. Use the following values for the Init Arguments for each instance of the servlet:
  - `InstallDir` — Directory where OpenEdge is installed.
  - `instanceName` — Name that you specified for the AIA instance in the `ubroker.properties` file.
  - `PropertyFileName` — Location of the `ubroker.properties` file. This would typically be `OpenEdge-Install-Directory/properties/ubroker.properties`.
  - Optionally set up a virtual path extension to point to AIA. For example, you could use a virtual path of `/aia/Aia`. Progress Software Corporation strongly recommends that you define a virtual path.

**Note:** The values you enter are case sensitive, so you must enter the values to meet the requirements of the platform you use.

- On UNIX only, you must add the following environment variables to the script that starts the JSE:
  - Add:
    
    ```
    DLC=OpenEdge-Install-Directory; export DLC
    ```
  - Add:
    
    ```
    WRKDIR=OpenEdge-working-directory; export WRKDIR
    ```

**Note:** This is the path to the work directory you specified when you installed OpenEdge.
Installing and configuring Web servers and Java servlet engines

- Add:

```
LD_LIBRARY_PATH = OpenEdge-Install-Directory/lib:$LD_LIBRARY_PATH;
export LD_LIBRARY_PATH
```

**Note:** Depending on your operating system, the library path variable might differ (for example, LIBPATH, SHLIB_PATH, and so on).

You must restart your JSE and Web server for these changes to take effect. After you have completed these steps, test the AIA configuration using a Web browser with the following URL:

**Syntax**

```
http://Host:Port/Path
```

**Host**

Identifies the machine where the Web server is running.

**Port**

Identifies the port number for the Web server.

**Path**

Identifies the virtual path and servlet name used to invoke the JSE and the AIA and must include the following information:

- A virtual path that the JSE has configured within the Web server to recognize and pass directly to the JSE instead of looking for that directory on the Web server

- Name of the Java servlet to be invoked by the JSE

The specification of the **Path** depends on your Web server and JSE. For more information, see the “Configuring AIA components” section on page 3–7 and the documentation for your Web server and servlet engine.

If you used all the recommended settings, the URL is:

```
http://Host:Port/aia/Aia
```

For instructions on installing and configuring a specific JSE, see the documentation for that JSE. For more specific information about configuring the AIA with JSEs, see the Release Notes and the Progress Knowledge Center on the Web (http://www.progress.com/support). For information about the configuration tasks you must complete to use the AIA with WebClient, see the Progress Knowledge Center on the Web (http://www.progress.com/support).
Enabling the Web server or JSE for SSL operation

Complete this procedure as part of the process of enabling the AIA for SSL communications.

To secure the Web server or JSE:

1. Send a request to a Certificate Authority for a digital certificate.
2. Receive the certificate from the Certificate Authority.
3. Install the certificate on the Web server.
4. Enable encryption on the Web server.

At the client machine, the Root Certification Authority (CA) certificate for all server certificates must be installed before HTTPS can be used. For WebClients and ABL clients, the Root CA certificates must be installed in the `OpenEdge-Install-Directory/certs` directory, where `OpenEdge-Install-Directory` identifies the directory where OpenEdge is installed. Root CA certificates for publicly available and well-known CAs, such as Verisign®, are automatically installed into the `OpenEdge-Install-Directory/certs` directory during the OpenEdge installation.

For more information about OpenEdge SSL support, see *OpenEdge Getting Started: Core Business Services* and the “Security considerations for AIA administration” section on page 3–2.
Configuring AIA components

You must configure AIA for your specific deployment. A running AIA instance is an extension of the JSE, and the JSE starts and stops the AIA instance when necessary. When the JSE starts an AIA instance on behalf of a client, the AIA instance reads the local ubroker.properties file to load its configuration information.

**Note:** The configuration information is loaded the first time that instance of an AIA is started, and the AIA continues to use that configuration information until it is stopped and restarted by the JSE.

This section describes how to configure an AIA instance by using OpenEdge Explorer or Progress Explorer and by editing the ubroker.properties file.

**General steps for using OpenEdge Explorer or Progress Explorer to configure an AIA instance**

Access OpenEdge Explorer or Progress Explorer from the OpenEdge program group in Windows. For more options, see OpenEdge Getting Started: Installation and Configuration.

To configure AIA instances:

1. Make sure that the AdminServer process is running on each of the following machines:
   - The machine where AIAs you want to configure are installed
   - The machine where all NameServers you want to configure are installed

In Windows, when you complete OpenEdge installation, the AdminServer automatically starts and is configured to auto-start whenever you reboot your system. You can also start the AdminServer (AdminService for OpenEdge) from the Windows Services applet available from the Control Panel. The AdminServer is running if you see both a java.exe and admsrvc.exe process listed in the Processes tab of the Task Manager.

On UNIX, use the proadsv command to start the AdminServer. To check whether the AdminServer is running, run the ps command to show the full command line for each process on the system and locate any jre commands in the list. The AdminServer process is running if you see a jre command with the arguments that correspond to those specified for jvmstart in the OpenEdge proadsv shell script located in OpenEdge-Install-Directory/bin.

For more information on using the PROADSV utility, see the “PROADSV” section on page B–19.

For more information on starting the AdminServer, see OpenEdge Getting Started: Installation and Configuration.

2. In OpenEdge Explorer or Progress Explorer, connect to the running AdminServer processes that you verified in Step 1. After you connect to a machine running an AdminServer process, its host name or IP address appears in the tool.

3. Locate the list showing all OpenEdge server products installed on the machine.
4. If you are using the NameServer, select the NameServer product where you want to configure NameServers and configure one or more such instances to support all AIA instances you want to configure. Note that for every AIA instance that you plan to run on a separate machine from its controlling NameServer, you must configure a remote NameServer instance on the same machine as the AIA. This remote NameServer instance must reference the host and port of the controlling NameServer for the AIA. For more information, see the chapter on configuring OpenEdge Unified Broker products in *OpenEdge Getting Started: Installation and Configuration*.

5. Select the AIA product where you want the AppServer instances to reside. If you are using the NameServer, configure the instances to register with the controlling NameServer (with or without load balancing). Specify any application service names required for clients to access the AIA and any other required configuration information. For more information, see the “Configuring an AIA with OpenEdge Explorer” section on page 3–8.

The sections that follow describe the basic steps for configuring each AIA instance. For more information, see the OpenEdge Explorer online help.

## Configuring an AIA with OpenEdge Explorer

To configure the AIA in Windows, you can use OpenEdge Explorer.

#### To define and configure an AIA instance:

1. Make sure the AdminServer is running on the host where you want to configure the AIA (see the “General steps for using OpenEdge Explorer or Progress Explorer to configure an AIA instance” section on page 3–7).

2. Open OpenEdge Explorer.

3. Connect to the AdminServer on your AIA host (see *OpenEdge Getting Started: Installation and Configuration*).

4. To either define a new AIA or modify an existing AIA, do one of the following:
   - To define a new AIA, select **AppServer Internet Adapter** folder in the tree view, and choose **Add Resource, New, AppServer Internet Adapter**. Enter a unique name for the AIA and click **Save**.
   - To modify an existing AIA configuration, expand the **AppServer Internet Adapter** folder in the tree view, select the AIA instance you want to modify, and choose **Configuration** on the right.

5. Select a property category by selecting a tab and set the properties as required. You can accept the default values, if they are appropriate for your application. You probably want to specify the properties under each category. See the online help for detailed information about the following categories:

   - **General** — Enter an **Idle connection timeout** value, in seconds, to specify the period of inactivity between the AIA and the client after which the AIA automatically disconnects from the AppServer or BrokerConnect. If you want this AIA instance to be HTTPS-enabled, check the box and enter the **Secure Port** number to specify where the AIA listens for HTTPS requests.
• **Controlling NameServer** — You must indicate whether you plan to use a NameServer to control AIA access. If so, check the **Register using NameServer** box and select a controlling NameServer from the list of NameServer instances that you have already configured; also, with the NameServer **Client Port** fields, enter minimum and maximum values (between 1024 and 65535) for the UDP port that the AIA instance can select to communicate with the NameServer (or enter **0** in both fields to allow random port selection within that range). If you are not using a NameServer, specify in the **AppServer host name and AppServer port** to handle client requests sent to this AIA instance.

• **Logging Setting** — You can set the following logging options: specify a different pathname from the default for the broker log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous broker log file; specify a comma-separated list of logging entry types to be included in the broker log file, choosing from the valid values listed in the Progress Explorer online help; set a file-size threshold that determines the point at which a new log file is created (0 = unlimited log file size); and specify the maximum number of broker log files to be kept (0 = unlimited number of log files retained). See *OpenEdge Development: Debugging and Troubleshooting* for detailed information on logging options.

• **SSL** — Check the **Enable SSL client connections** box if you want the AIA to be SSL-enabled; that is, to use SSL tunneling when connecting to the AppServer or BrokerConnect. If the AIA is SSL-enabled, you can optionally check the remaining boxes to disable host name verification and to prevent clients from requesting reuse of the session ID for successive connections to the same AppServer or BrokerConnect. (Leaving the **Disable SSL session Reuse** box unchecked does not guarantee that session IDs can be reused, because the server might disallow session reuse.)

• **Advanced Features** — To make it possible to run the AIA’s internal administrative commands from one or more IP addresses that you specify, check the **Internal administrative command** box. If this box is checked, you can maintain a list of authorized IP addresses. To add an address, type it in the **Authorized IP list** field.

### Configuring an AIA with the Progress Explorer

To configure the AIA in Windows, you can use Progress Explorer.

To define and configure an AIA instance:

1. Make sure the AdminServer is running on the host where you want to configure the AIA (see the “General steps for using OpenEdge Explorer or Progress Explorer to configure an AIA instance” section on page 3–7).

2. Open Progress Explorer.

3. Connect to the AdminServer on your AIA host (see *OpenEdge Getting Started: Installation and Configuration*).
4. To either define a new AIA or modify an existing AIA, do one of the following:

   • To define a new AIA, select the AIA folder in the Progress Explorer’s tree view, right-click, and choose New. Enter a unique name for the AIA and click OK. Then open the property editor for the new instance by selecting the instance, right-clicking, and choosing Properties.

   • To modify an existing AIA configuration, expand the AIA folder in the tree view, select the AIA instance you want to modify, and open the AIA property editor. The AIA instance property editor shows a tree view of property categories on the left and the properties for the selected category on the right.

5. Select a property category and set the properties as required. You can accept the default values, if they are appropriate for your application. You probably want to specify the properties under each category. See the Progress Explorer online help for detailed information about the following categories:

   • **General** — Enter an Idle Connection Timeout value, in seconds, to specify the period of inactivity between the AIA and the client after which the AIA automatically disconnects from the AppServer or BrokerConnect. If you want this AIA instance to be HTTPS-enabled, check the box and enter the Secure Port number to specify where the AIA listens for HTTPS requests.

   • **Controlling NameServer** — You must indicate whether you plan to use a NameServer to control AIA access. If so, check the Register With NameServer box and select a controlling NameServer from the list of NameServer instances that you have already configured; also, in the NameServer Client Port Range section, enter minimum and maximum values (between 1024 and 65535) for the UDP port that the AIA instance can select to communicate with the NameServer (or enter 0 in both fields to allow random port selection within that range). If you are not using a NameServer, specify in the AppServer Location section the host and port location of the AppServer that is to handle client requests sent to this AIA instance.

   • **Logging Setting** — You can set the following logging options: specify a different pathname from the default for the broker log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous broker log file; specify a comma-separated list of logging entry types to be included in the broker log file, choosing from the valid values listed in the Progress Explorer online help; set a file-size threshold that determines the point at which a new log file is created (0 = unlimited log file size); and specify the maximum number of broker log files to be kept (0 = unlimited number of log files retained). See OpenEdge Development: Debugging and Troubleshooting for detailed information on logging options.

   • **SSL** — Check the Enable SSL AppServer Connections box if you want the AIA to be SSL-enabled; that is, to use SSL tunneling when connecting to the AppServer or BrokerConnect. If the AIA is SSL-enabled, you can optionally check the remaining boxes to disable host name verification and to prevent clients from requesting reuse of the session ID for successive connections to the same AppServer or BrokerConnect. (Leaving the Disable SSL Session Reuse box unchecked does not guarantee that session IDs can be reused, because the server might disallow session reuse.)
• **Advanced Features** — To make it possible to run the AIA’s internal administrative commands from one or more IP addresses that you specify, check the **Internal Administrative Command** box. If this box is checked, you can maintain a list of authorized IP addresses. To add an address, type it in the **IP Address** field and click **Add**; to remove an address from the list, select it and click **Delete**.

### Configuring an AIA by editing the properties file

Although it is recommended that you use OpenEdge Explorer whenever possible, as an alternative you can edit the `ubroker.properties` file to configure the AIA. See the “Editing the properties file” section on page 2–24 for a description and guidelines for editing this file.

#### AIA entities in the properties file

The AIA configurations in `ubroker.properties` can include the entities listed in Table 3–1.

**Table 3–1: AIA configuration entity names**

<table>
<thead>
<tr>
<th>Configuration entity name</th>
<th>Configuration entity function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[AIA]</td>
<td>Defines default property settings for all instances of an AIA.</td>
</tr>
<tr>
<td>[AIA.product-instance-name]</td>
<td>Defines property settings for this instance of an AIA. The <code>ubroker.properties</code> file can contain several of these entities, each with a unique <code>product-instance-name</code>. For an AIA, <code>product-instance-name</code> is the name you specify for the AIA instance.</td>
</tr>
</tbody>
</table>

For complete definitions of all the properties and detailed information on how to set them, see the `ubroker.properties.README` file, as well as the comments included in the properties file itself. Both files reside in the `properties` directory.
Validating changes to the ubroker.properties file

If you use a tool other than OpenEdge Explorer to edit the values in the ubroker.properties file, you can use the AIACONFIG command to validate the changes made to the ubroker.properties file for an AIA instance.

The AIACONFIG command has the following syntax:

**Syntax**

```
aiaconfig [ [ -name AIA-instance-name ] [ -propfile path-to-properties-file ] [ -validate ] ] | -help
```

**Note:** If you upgrade to a new version of OpenEdge, you might want to retain changes made to the previous version’s ubroker.properties file. If that is the case, place the old properties file in the installed properties directory to replace the default. When starting the AdminServer for the first time, a properties file conversion utility automatically runs.

For more information on the AIACONFIG command, see the “AIACONFIG” section on page B–6.
Viewing AppServer Internet Adapter connection and configuration information

OpenEdge provides utilities to help you test and debug AIA using a browser. You can use your browser and URLs in the following form to view connection and configuration information about an AIA:

**Syntax**

```
http://Host:Port/Path
```

For more information on this URL, see the “Installing and configuring Web servers and Java servlet engines” section on page 3–4.

**Note:** By default, the allowAiaCmds property in the ubroker.properties file is turned off (set to 0) for security reasons. If you want to test the AIA using a browser, you must change the allowAiaCmds value to 1, and then stop and restart the JSE.

The **Path** used to view connection and configuration information depends on how you configure your Web server and Java servlet engine. In these examples, the **Path** was configured based on the default configuration used by OpenEdge. The first line verifies that AIA is configured correctly and is being successfully executed by the JSE. If you cannot run this example, there is a problem with the configuration. The second line provides status information about AIA, as shown:

```
http://Host:Port/aia/Aia
http://Host:Port/aia/Aia?GetServletStatus
```

**Connection status**

The connection status provides dynamic information about each client connection. This sample shows status information using the following URL:

```
http://starbuck:84/aia/Aia
```

Figure 3–1 shows connection information using the above URL.

**Figure 3–1:** AIA client connection information
Connection information provides you with information about that AIA configuration. If the AIA is configured correctly, the following information is displayed:

- The first line shows that the AIA servlet was accessed successfully.
- The second line, Connection Pool, shows the current and total connections to the AIA servlet since startup.
- The third line shows the current connection information including the thread number, the AppServer and IP address, total number of connections, and the date and time the servlet was last accessed.

**Configuration information**

Configuration information comes from the `ubroker.properties` file. This sample shows configuration information using the following URL:

```
http://starbuck:84/aia/Aia?GetServletStatus
```

Figure 3–2 shows AIA configuration information for the previous sample URL.
The configuration information includes the following:

- **ID** — An identification number used for tracking connections. A connection ID is assigned when the connection is made.

- **User Info** — Identifies the user. This information displays either just the user name, the user name and password, or the value **None**.

- **AppSvc** — Identifies the application service being used by the connection.

- **Host:Port** — Host and port of the client process using the connection.

- **#Pkt Sent** — Number of packets sent during a connection.

- **#Pkt Rcvd** — Number of packets received during a connection.

- **Connected** — Time the client connected.

- **Last Accessed** — Last time a request was sent or received.
Part III

Web Services Adapter Administration

Chapter 4, Configuring a Web Services Adapter Installation
Chapter 5, Managing the Web Services Adapter
Chapter 6, Deploying and Managing OpenEdge Web Services
Chapter 7, Web Services Adapter Security Configurations
Chapter 8, Using the WSA Management Utility (WSAMAN)
Configuring a Web Services Adapter
Installation

After OpenEdge is installed, you might have to perform one or more post-installation configuration tasks for the Web Services Adapter (WSA) in order to deploy OpenEdge Web services. You can configure the WSA at three different levels:

1. The Web application level of the Java servlet engine (JSE) where the WSA is installed. You can define multiple Web applications, each of which can run multiple instances of the WSA.

2. The WSA instance level of a single JSE Web application. Here, you can specify parameters that affect the JSE environment and execution of a single WSA instance.

3. The WSA instance within the OpenEdge environment. Here you can manage the OpenEdge side of WSA creation and execution, using the Unified Broker framework.

These tasks are described in the following sections:

- Moving the WSA sample Web application
- Configuring the JSE to recognize the WSA
- Tuning the JSE to handle large Web service messages
- Modifying the WSA’s default Web service security settings
- Enabling the WSA for HTTPS client connections
- Installing WSA on a DMZ server
Moving the WSA sample Web application

The OpenEdge Web Services Adapter (WSA) component comes with a sample WSA Web application, which is installed, by default, within the OpenEdge hierarchy. This works in development environments. But in production environments, where OpenEdge might be uninstalled and reinstalled frequently, the sample WSA Web application might be deleted. For this reason, Progress Software Corporation recommends that in production environments, you copy or move the WSA sample Web application to a disk location outside the OpenEdge hierarchy—perhaps to the Web application directory of the Java servlet engine (JSE) where the WSA runs.

To copy the WSA sample Web application:

1. Change the directory to OpenEdge-Install-Directory/servlets.
2. Copy the entire WSA subdirectory tree to its final destination. Use the command for your operating system, as shown in Table 4–1.

<table>
<thead>
<tr>
<th>On this operating system</th>
<th>Use this command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>xcopy /E /I ./wsa destination-directory</td>
</tr>
<tr>
<td>UNIX</td>
<td>cp -r -p ./wsa destination-directory</td>
</tr>
</tbody>
</table>

3. Update your JSE configuration files to refer to the new WSA sample Web application directory. For more information, see the “Configuring the JSE to recognize the WSA” section on page 4–3. In general, you need to restart the JSE (or Web server) for the new configuration settings to take effect.

4. If you plan to deploy Web services to WSA instances created in the new sample Web application directory, destination-directory/wsa, you can proceed with deployment immediately after completing Step 3. In this case all Web services are deployed to a WSA instance directory within destination-directory/wsa. However, if you want to specify a separate Web service deployment directory for any WSA instance created for the Web application, you must update the Web application descriptor file to specify a new deployment directory for that instance. For more information, see Chapter 6, “Deploying and Managing OpenEdge Web Services.”
Configuring the JSE to recognize the WSA

The WSA component runs as a Web application within a JSE. Therefore, you must inform the JSE that the WSA component is one of its Web applications. You must also update this information if you move the Web application directory for a given Web application (see the “Moving the WSA sample Web application” section on page 4–2). To configure the JSE to recognize the WSA component, and for more information on installing Web applications from an expanded source-directory tree, see your JSE documentation. Otherwise, for information on configuring the WSA as a Web application, see Chapter 5, “Managing the Web Services Adapter.”
Tuning the JSE to handle large Web service messages

If your Web service attempts to send a large amount of data to a client (for example, 10 MB in a single SOAP message), the JSE can fail to send the SOAP response message for the client request, effectively causing the Web service to hang. To solve or prevent this problem, you can increase the size of the memory allocation pool in the Java virtual machine (JVM) for the Java servlet container that runs the WSA.

To increase this allocation pool, set the `-Xmx` JVM startup option to a higher value. For example, you might set the value to 512 MB by specifying this option as `-Xmx512m`.

For more information on this option and JVM memory management, see the documentation on tuning garbage collection with the JVM available at the Java Web site (http://Java.sun.com).
Modifying the WSA’s default Web service security settings

When you install the WSA component, its security settings are preset, but modifiable. These settings affect:

- WSA administrators
- Client developers
- Web service application users

For more information on these security settings, including instructions for modifying them, see Chapter 7, “Web Services Adapter Security Configurations.”

The WSA supports SSL communications with the AppServer with the use of a secure protocol. You control the default protocol for services deployed to a WSA instance by setting the value of the appServiceProtocol property in the default.props file. To enable SSL tunneling, you specify either AppServerS (if the service is registered with a NameServer) or AppServerDCS (for direct connection to the AppServer) as the protocol. For more information, see the “Security considerations for Web service administration” section on page 6-2.
Enabling the WSA for HTTPS client connections

OpenEdge supports Secure Socket Layer (SSL)-enabled communications between the Web server where the WSA runs and a Web service client. To achieve this, you must enable the WSA for HTTPS (SSL) connections.

**Note:** The procedure that follows pertains to the security of communication between the client application and the WSA. To enable SSL communication between the WSA and the AppServer, you must obtain and install public key certificates for the WSA host machine and complete separate configuration procedures for each deployed service and for the AppServer. See the “Security considerations for Web service administration” section on page 6-2 for more information.

To enable the WSA for SSL:

1. Obtain a private key and a Web server digital certificate.

2. Enable and configure the Web server for SSL support.

   This includes installing the Web server digital certificate in the Web server. For more information, see the Web server documentation.

3. Using a text editor, modify the `web.xml` file for the WSA as follows:

   In the `<security-context>` element for WSA administration, located in the `<transport-guarantee>` element, change the value from `NONE` to `CONFIDENTIAL`.

   **Note:** For more information on `web.xml`, see Chapter 5, “WSA administration architecture.”

4. Using a text editor, edit the `ubroker.properties` file to set the WSA URL (value of the `wsaUrl` property) to use the HTTPS protocol (instead of HTTP).
Installing WSA on a DMZ server

A DMZ server is a server that might not allow you to open untrusted network ports. Since the AdminServer uses ports which might be interpreted as untrusted, the standard WSA configuration might run on such a server. The WSA supports a “remote” configuration to work around this issue. The remote configuration installs only enough of OpenEdge on the remote server to run the WSA. This installation includes the WSA Java Servlet and the files used to support its local configuration.

The remote WSA uses a ubroker.properties file with a limited set of static startup parameters. You must manually edit this file to change the WSA instance’s static configuration properties.

You can continue to administer the remote WSA’s dynamic run-time properties and deploy and manage SOAP services by mapping it to an AdminServer on the Intranet behind the DMZ server. The Intranet AdminServer uses HTTP/S to pass WSA administration operations as SOAP messages to the WSA’s Administration service.

To run a remote WSA configuration:

1. Install the remote WSA on the DMZ server. Manually edit the static startup parameters in its ubroker.properties file and install the WSA as a Web application in the local Web server’s JSE.

2. Start the WSA the Web server’s JSE and verify that a browser can access its URL from the Internet.

3. Connect to an AdminServer on the Intranet and configure a “remote” WSA instance using OpenEdge Explorer or the WSMAN utility by specifying the remote WSA’s URL.

   The Intranet AdminServer creates a WSA mapping entry in its local ubroker.properties file with the URL of the remote WSA’s administration SOAP service.

4. Use OpenEdge Explorer or the WSMAN utility to manage the “remote” WSA’s dynamic run-time properties and deploy and manage the WSA’s SOAP services.

   OpenEdge Explorer connects to the Intranet AdminServer where you mapped the WSA and then connects to the remote WSA’s SOAP administration service. When OpenEdge Explorer sends an operation to the Intranet AdminServer, the AdminServer turns it into a SOAP message and forwards it to the WSA’s SOAP administration service which executes the operation.
Managing the Web Services Adapter

The Web Services Adapter (WSA) is a Java servlet running within a Java servlet engine (JSE). You can manage the WSA using the Unified Broker framework, including OpenEdge Explorer, WSAMAN utility, and the WSACONFIG utility. The WSA serves as the gateway and management engine for your Web service. It provides user access to the Web Services Description Language (WSDL) file, supports all required Web service administration, and manages the exchange of Web service Simple Object Access Protocol (SOAP) requests and responses between the Web service and Web service clients at run time.

You must configure the WSA and create and configure WSA instances before you can deploy Web services to them. This chapter contains the following sections:

- WSA administration architecture
- Defining a Web application for the WSA
- Creating one or more WSA instances
- Starting and testing a WSA instance
- Configuring and managing a WSA instance

The WSA includes extensive security features, which this chapter introduces, but for more detailed information on WSA security, see Chapter 7, “Web Services Adapter Security Configurations.”
WSA administration architecture

To configure the WSA, you must understand its architecture, which is explained in the following sections:

- WSA administration prerequisites
- Web server and JSE configurations
- Constructing URLs
- WSA as a JSE Web application

**Note:** For a complete overview of OpenEdge Web services architecture and how the WSA supports Web services, see *OpenEdge Getting Started: Application and Integration Services*.

WSA administration prerequisites

You use OpenEdge Explorer or WSAMAN command-line utility to manage the WSA. However, before you can manage the WSA, you must complete the following installation requirements:

- On the system where the JSE is running, install the WSA component of OpenEdge.
- On the system where the AdminServer to administer the WSA is running, install the Web Services Admin Enabler component. The AdminServer can reside on the same machine as the WSA (local) or on a separate machine (remote).

For more information on installing these components and related products, see *OpenEdge Getting Started: Installation and Configuration*. 
Figure 5-1 shows the components used to configure and manage a WSA in its environment.

**Web server and JSE configurations**

As shown in Figure 5-1, the WSA is implemented as a Java servlet that runs in the context of a JSE as a Web application. The JSE can run stand-alone (that is, with an integrated Web server), or in the context of an optional, separate Web server (such as Apache or IIS).

When the WSA runs within the context of a JSE or Web server, several WSA instances can run within the context of the WSA Web application (such as wsa1 and wsa2 shown in Figure 5-1). In addition to the WSA Web application other Web applications can run within the JSE, including additional WSA instances.

The configuration of a Web server-based WSA is as follows:

```
Web server ( root )
  JSE ( web-server-context )
    WSA Web application ( wsa-webapp-context )
      WSA instance ( wsa-instance )
        ...
        ...
    Non-WSA Web application
      ...
      ...
    Non-JSE Web resource, such as a static html page
      ...
      ...
```
In the Web server-based WSA configuration:

1. There is a separate Web server.
2. The Web server comprises one or more JSEs (\textit{web-server-context}) and one or more non-JSE Web resources, such as static HTML pages.
3. Each JSE comprises one or more WSA Web applications (\textit{wsa-webapp-context}) and one or more non-WSA Web applications.
4. Each WSA Web application comprises one or more WSA instances (\textit{wsa-instance}).

If the JSE runs stand-alone (that is, with an integrated Web server), the JSE-based WSA configuration is as follows:

\begin{verbatim}
Stand-alone JSE (root)
   WSA Web application (wsa-webapp-context)
      WSA instance (wsa-instance)
      ...
      ...
      Non-WSA Web application
      ...
\end{verbatim}

In both these configurations, \textit{web-server-context}, \textit{wsa-webapp-context}, and \textit{wsa-instance} appear in bold because they are components of the URL for accessing the WSA.

**Constructing URLs**

In this manual, the URL path to a specific WSA instance is referred to as \textit{wsa-root-url}. When you install the WSA, the installation procedure prompts you for a \textit{wsa-root-url} value for the sample WSA instance (wsa1). Thereafter, every time you create a new WSA instance you must provide a new \textit{wsa-root-url} value for that instance. The value is stored in the \textit{wsaUrl} property in \texttt{ubroker.properties}.

The \textit{wsa-root-url} has the following syntax:

**Syntax**

\begin{verbatim}
http[s]://host:port[/web-server-context]/wsa-webapp-context/wsa-instance
\end{verbatim}

**Note:** If the WSA URL host is named localhost, it must be changed to the name of the host machine when Web services are deployed for network access.

In the previous syntax, use the optional element, \textit{/web-server-context}, if the WSA runs in a JSE that runs in a separate Web server. Omit the optional element if the WSA runs in a JSE that runs stand-alone.
In order to create a WSA instance, you need to construct the URL for the `wsa-root-url`. The URL components in the syntaxes each describe a specific part of the WSA instance configuration, as shown in Table 5–1.

### Table 5–1: URL components

<table>
<thead>
<tr>
<th>This URL component</th>
<th>Which is specified in . . .</th>
<th>Names . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>web-server-context</code></td>
<td>The Web server configuration file</td>
<td>The connection between the JSE and the Web server (if any)</td>
</tr>
<tr>
<td><code>wsa-webapp-context</code></td>
<td>The JSE configuration file</td>
<td>A particular Web application within the JSE context</td>
</tr>
<tr>
<td><code>wsa-instance</code></td>
<td>The WSA Web application descriptor file</td>
<td>A unique WSA instance</td>
</tr>
</tbody>
</table>

Typically, `web-server-context`, `wsa-webapp-context`, and `wsa-instance` are assigned aliases (actual values) by providers—the people who are responsible for the hardware and software in the OpenEdge WSA environment. Aliases typically do not consist of physical names related to the system or to file paths, but rather names meaningful to providers.

For example, consider the sample Web service system elements and aliases in Table 5–2.

### Table 5–2: Sample aliases for URL components

<table>
<thead>
<tr>
<th>URL component</th>
<th>Sample alias</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>web-server-context</code></td>
<td>bedrock</td>
</tr>
<tr>
<td><code>wsa-webapp-context</code></td>
<td>quarry</td>
</tr>
<tr>
<td><code>wsa-instance</code></td>
<td>fred</td>
</tr>
</tbody>
</table>

Given the aliases in Table 5–2, the following are true:

- The WSA running in the context of the JSE running in the context of a separate Web server has the following URL:

  http://servicehost:80/bェドッグ/quarry/fred

- The WSA running in the context of the JSE running stand-alone has the following URL:

  http://servicehost:8080/quarry/fred
Any URL subpaths specified by the client that lie beyond \texttt{wsa-root-url} are delivered to the WSA instance for it to interpret and use. For example, to access the WSDL documents associated with a particular WSA instance, append the /wsdl subpath to \texttt{wsa-root-url}, as shown in bold:

\begin{verbatim}
http://servicehost:80/bedrock/quarry/fred/wsdl
\end{verbatim}

Given this URL, an HTTP message travels through the Web server, through the JSE, then to the WSA instance, fred. Then, the WSA instance is passed the relative path, /wsdl, which tells it to perform a WSDL-related operation.

**WSA as a JSE Web application**

To configure a WSA, it is also necessary to understand the architecture of the WSA as a JSE Web application. The WSA Web application consists of a set of directories and files. Figure 5–2 shows the directory structure.

![Directory structure of the WSA as a JSE Web application](image)

The WSA Web application consists of a subdirectory named \texttt{WEB-INF} (the name is case sensitive) that contains the Web application descriptor file (\texttt{web.xml}) and two subdirectories for holding Java class files and Java jar files. The WSA Web application does not contain any class files, but does include several jar files. These form a single Web application that can contain one or more WSA instances. A JSE can contain multiple WSA Web applications, and each Web application can contains multiple WSA instances.

Configuring a WSA Web application involves editing its \texttt{web.xml} file, an XML-based file that must conform to the JSE version’s document type definition (DTD) published by Sun Microsystems, Inc. You might have to manually edit \texttt{web.xml} to set WSA instance security or to add a WSA instance to the WSA Web application. For a full description of the \texttt{web.xml} file, see Sun’s JSE specifications.
Defining a Web application for the WSA

You must define a Web application to allow the JSE to locate the WSA.

Note: The steps below assume that the sample WSA Web application provided with OpenEdge is installed in the OpenEdge-Install-Directory/servlets/wsa directory.

To define a Web application for the WSA:

1. If you are running a production server, or if you do not want to lose your deployed Web services when installing another version of OpenEdge, copy the entire sample WSA Web application directory tree to another disk location, outside the OpenEdge installation directory. For an example of how this might be done see the information on copying the WSA sample Web application in Chapter 4, “Configuring a Web Services Adapter Installation.”

2. Configure the JSE to recognize the WSA Web application. Depending upon the JSE, you might have to edit one of its configuration files or install the WSA Web application by using a browser-based administration interface. In addition, each JSE typically has a directory where a Web application resides. See your JSE vendor’s documentation for the specific directory name and a discussion of Web applications.

Note: If a Web application installation procedure requests the path to the Web application deployment file, enter the path to the WSA Web application. For example, the Web application deployment path created during OpenEdge installation is OpenEdge-Install-Directory/servlets/wsa.

3. If you chose to enable security when you installed the WSA Web application sample, you must now define the WSA administration user names and roles in the JSE. For more information, see your JSE vendor’s documentation. The WSA comes preconfigured to use the role definitions PSCAdmin and PSC0per (case-sensitive). With security enabled, the WSA requires the JSE to authenticate and authorize a user before passing the HTTP request to the JSE for handling. For more information on WSA security, see Chapter 7, “Web Services Adapter Security Configurations.”
Creating one or more WSA instances

This section describes how to create a WSA instance. This section also describes how to create multiple WSA instances and why you might want to do this.

Creating a WSA instance

You use OpenEdge Explorer to create a WSA instance.

To create a WSA instance:

1. In OpenEdge Explorer’s left pane, right-click on the Web Services Adapter, and click Add Resource, New, Web Service Adapter.

2. Supply the new WSA instance’s name and URL and click Save. When the left pane appears, it now contains the new WSA instance.

3. Using the tabbed interface, configure the desired properties.

4. If the WSA instance is remote, perform the following additional steps on the system where the WSA is installed:
   a. In the ubroker.properties file, copy and rename the section corresponding to the sample WSA instance provided (wsa1) to a new section—in essence, cloning wsa1’s section to a new section.
   b. Then, edit the properties in the new WSA instance’s section as desired.

5. In the web.xml file, use a text editor (or whatever tool your JSE provides), to copy and rename the section corresponding to the sample WSA instance provided (wsa1) to a new section—in essence, cloning wsa1’s section to a new section.

Then, check and perhaps modify, the web.xml items listed in Table 5–3.

Table 5–3: web.xml file items to check

<table>
<thead>
<tr>
<th>Check this attribute . . .</th>
<th>Which indicates . . .</th>
<th>And whose default is . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;init-param&gt;</td>
<td>The location on disk of the OpenEdge installation directory</td>
<td>C:\Progress\OpenEdge</td>
</tr>
<tr>
<td>&lt;param-name&gt;</td>
<td>InstallDir</td>
<td></td>
</tr>
<tr>
<td>&lt;param-value&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| <init-param>              | The name of the WSA instance | wsa1                        |
| <param-name>              | instanceName              |                            |
| <param-value>             |                        |                            |

| <init-param>              | The location of the ubroker.properties file | C:\Progress\OpenEdge\ubroker.properties |
| <param-name>              | propertyFileName           | Note: It need not reside in InstallDir. |
| <param-value>             |                        |                            |

| <servlet-mapping>         | The name of the WSA servlet | wsa1_servlet               |
| <servlet-name>            |                        |                            |
Creating one or more WSA instances

Progress Software Corporation recommends that you use names consistently when you name a remote WSA instance and when you edit entries corresponding to the remote WSA instance in the `ubroker.properties` and `web.xml` files.

## Creating multiple WSA instances

You can create multiple WSA instances. You might do this to support different versions of the Web service or different levels of security. Suppose you have several AppServer applications to deploy as Web services, and you want only a particular group of users to have access to each Web service. You can accomplish this by creating a WSA instance for each group of users, then deploying to each WSA instance the Web services for that group. For more information on this and other security techniques, see Chapter 7, “Web Services Adapter Security Configurations.”

<table>
<thead>
<tr>
<th>Check this attribute . . .</th>
<th>Which indicates . . .</th>
<th>And whose default is . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;servlet-mapping&gt;</code></td>
<td>The URL pattern of the WSA servlet</td>
<td><code>/wsa1/*</code></td>
</tr>
<tr>
<td><code>&lt;security-constraint&gt;</code></td>
<td>The role-name of the default WSA administrator</td>
<td>PSCAdmin</td>
</tr>
<tr>
<td><code>&lt;security-constraint&gt;</code></td>
<td>The role-name of the default WSA operator</td>
<td>PSCOper</td>
</tr>
</tbody>
</table>
| `<user-data-constraint>` | The type of connection for accessing WSA administration, WSDL, and Web service applications | CONFIDENTIAL (if you are using SSL)  
 NONE (if you are not using SSL) |
Starting and testing a WSA instance

Once you create WSA instances you can start and test them.

Starting the WSA and its instances

Anytime that you edit the ubroker.properties or web.xml file for a WSA and its instances, you must restart the WSA.

To start and run the WSA:

1. Ensure that your JSE is running.
2. Following the instructions for your specific JSE, start the WSA as a Web application. See your JSE vendor documentation on starting and running a Web application.

**Note:** Depending on the behavior of a specific JSE, starting the JSE might automatically start the WSA.

Testing a WSA instance

After you create and start a WSA instance, you can test it.

To test a WSA instance:

1. Confirm that the WSA instance started correctly by checking for the presence of the files and directories in the following table:

<table>
<thead>
<tr>
<th>Check for the WSA instance’s . . .</th>
<th>The syntax is . . .</th>
<th>The default is . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log file (specified in the WSA instance’s logFile property)</td>
<td>OpenEdge-Work-Directory/wsa-instance-name.wsa.log</td>
<td>OpenEdge-Work-Directory/wsa1.wsa.log</td>
</tr>
<tr>
<td>Directory for holding deployed Web service applications</td>
<td>.../wsa-app/wsa-instance-name</td>
<td>OpenEdge-Install-Directory/servlets/wsa/wsa1/</td>
</tr>
<tr>
<td>Default Web service properties (default.props) file</td>
<td>.../wsa-app/wsa-instance-name/default.props</td>
<td>OpenEdge-Install-Directory/servlets/wsa/wsa1/default.props</td>
</tr>
</tbody>
</table>
2. Confirm that the WSA is responding by pointing a Web browser at the WSA instance’s URL (wsa-root-url).

For example, for JSE running stand-alone, use the following URL:

http://servicehost:8080/wsa/wsa1

The test is successful if you receive a **WSA Web Services OK** HTML page.
Configuring and managing a WSA instance

To configure and manage a WSA instance, set its properties and monitor its statistics.

Properties of a WSA instance

When you create a WSA instance, it is assigned a collection of startup properties that reside in the `ubroker.properties` file associated with the WSA instance. These properties include options for logging, debugging, and tuning. You can modify the startup properties of a WSA instance anytime after creating it using OpenEdge Explorer, or by manually editing the file.

Table 5–4 lists, by category, the properties of a WSA instance that reside in the `ubroker.properties` file. For more information on these properties, see the OpenEdge Explorer online help.

Table 5–4: `ubroker.properties` properties of a WSA instance (1 of 2)

<table>
<thead>
<tr>
<th>Category</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Web Services Adapter Location</td>
</tr>
<tr>
<td></td>
<td>URL</td>
</tr>
<tr>
<td></td>
<td>Web Service Protocol Type</td>
</tr>
<tr>
<td>Proxy Server Setting</td>
<td>Host Name</td>
</tr>
<tr>
<td></td>
<td>Port Number</td>
</tr>
<tr>
<td></td>
<td>Username</td>
</tr>
<tr>
<td></td>
<td>Password</td>
</tr>
<tr>
<td>WSDL</td>
<td>Enable WSDL Retrieval</td>
</tr>
<tr>
<td></td>
<td>WSDL Inaccessible Page</td>
</tr>
<tr>
<td></td>
<td>Enable WSDL Listing Retrieval</td>
</tr>
<tr>
<td></td>
<td>WSDL Listing Page</td>
</tr>
<tr>
<td>Logging</td>
<td>Working Directory</td>
</tr>
<tr>
<td></td>
<td>Log Filename</td>
</tr>
<tr>
<td></td>
<td>Web Services Adapter Logging Level</td>
</tr>
<tr>
<td></td>
<td>Append to Web Services Adapter Log File</td>
</tr>
<tr>
<td></td>
<td>Log Message Threshold</td>
</tr>
</tbody>
</table>
To modify most startup properties of a WSA instance, you must stop and start the JSE associated with the WSA instance. However, the following WSA properties can be modified at run time (while the WSA is running):

- debugClients
- enableWsdl
- enableWsdlListings
- loggingLevel
- logMsgThreshold
- webAppEnabled

Any modifications to these properties take affect immediately, but do not remain once the WSA is restarted. Changing these properties at run time does not save the new properties values. To make the run-time properties changes persistent you must also change the values of the startup properties.

To modify the run-time properties of a WSA instance, you can use OpenEdge Explorer or the WSAMAN utility. To modify the startup properties of a WSA instance, use OpenEdge Explorer. For more information on OpenEdge Explorer, see the online help. For more information on the WSAMAN utility, see Chapter 8, “Using the WSA Management Utility (WSAMAN)” and Appendix B, “Command and Utility Reference.”
At times, you might find that in order to update a WSA configuration, you must edit the
ubroker.properties file by hand using a text editor, especially when you need to update the
WSA security configuration. When you complete a manual update of the ubroker.properties
file, use the WSACONFIG utility to verify the integrity of the WSA configurations in the file.

**Note:** You can also use the mergeprop utility installed with OpenEdge to manually edit the
ubroker.properties file. For information on using mergeprop, see *OpenEdge
Getting Started: Installation and Configuration.*

The WSACONFIG utility displays the property settings associated with WSA configuration, and
checks that the syntax and values are valid. You must run the WSACONFIG utility locally on the
machine on which the WSA is running. The utility does not run across the network.

This is the syntax used to invoke the WSACONFIG utility:

**Syntax**

```bash
wsaconfig [-name WSA-instance-name] [-propfile path-to-properties-file] [-validate] [-help]
```

For more information on the WSACONFIG utility, see the “WSACONFIG” section on page B–20.

For example, the following command validates the syntax and views the configurations of all
WSA instances defined within the test.properties file located in the current working
directory:

```bash
wsaconfig -propfile test.properties -validate
```

**Statistics of a WSA instance**

When you create a WSA instance, statistics associated with the WSA instance are maintained.
That is, statistics are kept for each WSA instance and each Web service deployed to a WSA
instance. By inspecting and resetting these statistics, you can monitor and tune the WSA
instance. Table 5–5 lists the statistics of a WSA instance.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of SOAP requests</td>
<td>Requests for Web service applications</td>
</tr>
<tr>
<td>Number of Active requests</td>
<td>Requests queued and being acted upon</td>
</tr>
<tr>
<td>Number of HTTP requests</td>
<td>Total requests received from the HTTP listener, including administrative, WSDL, and Web service requests</td>
</tr>
<tr>
<td>Number of WSDL requests</td>
<td>Requests for WSDL documents</td>
</tr>
</tbody>
</table>
Configuring the WSA for Web services

Additional configuration settings exist for each WSA instance that affect the deployment of Web services. These include changing the default values for the properties of Web services that you deploy to a WSA instance.

When you first run a WSA instance, it creates a default.props file that contains a collection of properties assigned to each Web service deployed to the WSA instance. For additional information about the default values for Web services properties and how to change them, see Chapter 6, “Deploying and Managing OpenEdge Web Services.”

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of SOAP Faults</td>
<td>Error returns from Web service applications</td>
</tr>
<tr>
<td>Number of services disabled</td>
<td>Web services deployed to this WSA that are disabled from client access</td>
</tr>
<tr>
<td>Number of errors</td>
<td>Total errors returned by the WSA, with error counts broken out at the bottom of the list for each of several error categories when total errors are greater than 0</td>
</tr>
</tbody>
</table>
After the developer uses ProxyGen to define a Web service and create a Web Service Mapping (WSM) file, and after you create a Web Services Adapter (WSA) instance, you initialize the WSA instance’s default Web service properties, deploy the Web service to the WSA instance, enable the Web service, and finally configure the Web service.

The following sections in this chapter describes how to deploy Web service using the WSA:

- Initializing a WSA instance’s default Web service properties
- Deploying a Web service
- Changing the Web service deployment directory
- Enabling a Web service
- Administering a deployed Web service
- Web service file management for deployed Web services
- Monitoring and tuning Web services
- Typical Web service administration scenarios

For information on using ProxyGen to generate a WSM file for Web service deployment, see *OpenEdge Development: Open Client Introduction and Programming*. 
Initializing a WSA instance’s default Web service properties

Each WSA instance has a set of default Web service properties, which reside in the `default.props` file associated with the WSA instance. You can modify these, using OpenEdge Explorer or the WSAMAN utility. Table 6–1 lists some of these default Web services properties.

Table 6–1: Setting the WSA instance’s default Web service properties

<table>
<thead>
<tr>
<th>Set this property . . .</th>
<th>To . . .</th>
<th>The default is . . .</th>
</tr>
</thead>
</table>
| appServiceProtocol     | AppServer (if the AppServer configuration includes a NameServer and does not require SSL connections).
                                     AppServerS (if the AppServer configuration includes a NameServer and requires SSL connections).
                                     AppServerDC (if the AppServer configuration does not include a NameServer and does not require SSL connections).
                                     AppServerDCS (if the AppServer configuration does not include a NameServer and requires SSL connections). | AppServer |
| appServiceHost         | The name of the NameServer system or AppServer system, depending on the setting of appServiceProtocol. | localhost |
| appServicePort         | The TCP/IP port number, which depends on whether the appServiceHost property specifies a NameServer or AppServer system. | 5162 |
| appServiceName         | The name of the default AppServer service. | asbroker1 |
| serviceLoggingLevel    | A value between 1 and 4 to identify the logging level for Web services applications. The greater the value, the more logging occurs. | 2 |
| serviceAvailable       | 1 to allow client access to enabled Web service applications. Set to 0 to prevent client access. | 0 |

For more information on modifying the default Web service properties of a WSA instance, including a full list of default properties, see the OpenEdge Explorer online help, or see Chapter 8, “Using the WSA Management Utility (WSAMAN)” and Appendix B, “Command and Utility Reference.”

Security considerations for Web service administration

The WSA acts as an intermediary between the AppServer and clients that access the service over the Internet. Thus, an application session involves two distinct connections, each of which is configured separately with respect to security.
The first connection is Internet-based between the WSA and the client. See the “Enabling the WSA for HTTPS client connections” section on page 4–6 for information about making this connection secure. In brief, the following conditions must be met:

- The client must use HTTPS protocol to send requests
- The WSA must be HTTPS-enabled; that is, it must be configured to accept HTTPS requests from clients (via the JSE or Web server)
- A private key and a Web server digital certificate must be installed on the Web server, and the Web server must be configured for SSL support

The second connection is via AppServer protocol between the deployed service and the AppServer. For this connection to be secure, the following conditions must be met:

- You must obtain and install public key certificates for the WSA host machine.
- The service must send SSL requests to the AppServer that is to process the client requests. To configure the service to send SSL requests, you set the value of the appServiceProtocol property to AppServerS or AppServerDCS. You set this property, either for a specific service or as the default for services deployed to a given WSA instance, by using OpenEdge Explorer or by manually editing the WebServiceFriendlyName.props file or the default.props file. (Note that this property applies to deployed services, not to the WSA itself; for more information on configuring WSA security, see Chapter 7, “Web Services Adapter Security Configurations.”)
- The AppServer must be SSL-enabled, meaning that it accepts SSL requests from the WSA (or other clients). You set the property sslEnable=1 by checking the Enable SSL Client Connections box in the SSL General properties category in OpenEdge Explorer, or by manually editing the ubroker.properties file. You must also obtain and install a server private key and public key certificate and set additional SSL server properties. See the “SSL-enabled AppServer operation” section on page 2–8 for more information.

For details on SSL support in OpenEdge, including configuring and operating a Web service as a client of an SSL-enabled AppServer, see OpenEdge Getting Started: Core Business Services.

**SSL-related service properties**

You can set the following properties, either as defaults for services deployed to a given WSA instance or as properties of a specific service:

- **appServiceProtocol** — Assigns a value of AppServerS or AppServerDCS to support SSL communication with the AppServer
- **noHostVerify** — Controls whether the WSA compares the host name of the connecting AppServer with the Common Name specified in the server digital certificate
- **noSessionReuse** — Controls whether the service requests reuse of the session ID for successive connections to the same AppServer

For more information about these and other service properties, see Appendix A, “Reference to OpenEdge Web Service Properties.”
Deploying a Web service

You can use OpenEdge Explorer or the WSAMAN utility’s deploy function to deploy a Web service. This section describes deploying a Web service with OpenEdge Explorer. For more information on using WSAMAN to deploy a Web service, see Chapter 8, “Using the WSA Management Utility (WSAMAN)” and Appendix B, “Command and Utility Reference.”

See OpenEdge Management and OpenEdge Explorer: Getting Started for complete information on deploying a web service using OpenEdge Explorer. You can also access this information from the online help.

Versioning Web Services

You can use the friendly name and target namespace to support versioning for a Web service. This is useful when you want to deploy a Web service for more than one SOAP format (WSDL style/use), to use different AppServers, or to generally deploy an updated version of the Web service.

Progress Software Corporation recommends putting version information in the target namespace to identify multiple versions of a Web service. To support multiple SOAP formats, either deploy each SOAP format to a different WSA instance or indicate the SOAP format in the friendly name and target namespace.

For example, Table 6–2 shows how you might represent the first version of the Web service.

Table 6–2: Web service order info version 1

<table>
<thead>
<tr>
<th>Deployment information</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>OrderInfo-RpcEncoded-0001</td>
</tr>
<tr>
<td>Web Service Namespace</td>
<td>(one of these values)</td>
</tr>
<tr>
<td></td>
<td>urn:your-domain-name:OrderInfo:RpcEncoded:0001</td>
</tr>
<tr>
<td></td>
<td><a href="http://your-domain-name/OrderInfo/RpcEncoded/0001">http://your-domain-name/OrderInfo/RpcEncoded/0001</a></td>
</tr>
</tbody>
</table>

Table 6–3 shows how you might represent the second version of the Web service.

Table 6–3: Web service order info version 2

<table>
<thead>
<tr>
<th>Deployment information</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>OrderInfo-RpcEncoded-0002</td>
</tr>
<tr>
<td>Web Service Namespace</td>
<td>(one of these values)</td>
</tr>
<tr>
<td></td>
<td>urn:your-domain-name:OrderInfo:RpcEncoded:0002</td>
</tr>
<tr>
<td></td>
<td><a href="http://your-domain-name/OrderInfo/RpcEncoded/0002">http://your-domain-name/OrderInfo/RpcEncoded/0002</a></td>
</tr>
</tbody>
</table>
After deploying Web services

After deployment, in OpenEdge Explorer, you will notice an icon for the Web services under the Web Services folder for the WSA instance. Also, the files listed in Table 6–4 are created in the WSA instance directory.

**Table 6–4: File created when a Web service is deployed**

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>friendlyname.props</td>
<td>An XML file containing the current Web service property settings (initially set from default.props)</td>
</tr>
<tr>
<td>friendlyname.wsad</td>
<td>The Web Service Application Descriptor (WSAD) XML file that defines the Web service to the WSA instance</td>
</tr>
<tr>
<td>friendlyname.wsd1</td>
<td>The WSDL XML file that defines the Web service to potential Web service clients</td>
</tr>
</tbody>
</table>

During development, you can change the definition and deployment information of a deployed Web service using a Web service update. You can use OpenEdge Explorer or WSAMAN to do this. However, once deployed and enabled for client access under production conditions, avoid making any changes to this information, as client implementations depend on its stability.

To make the same Web service available using different information after production deployment (for example, to add a new operation or use a different SOAP Message Style), deploy a new Web service with the new information by using a different Web service name and target namespace at the same WSA instance (similar to versioning a Web service), or by deploying the Web service to a different WSA instance.

However, you can always change the run-time properties of a deployed Web service at any time, but most properties require that you first disable the Web service. For more information, see the OpenEdge Explorer online help, Chapter 8, “Using the WSA Management Utility (WSAMAN),” Appendix B, “Command and Utility Reference,” and Appendix A, “Reference to OpenEdge Web Service Properties.”
Changing the Web service deployment directory

As described in Chapter 4, “Configuring a Web Services Adapter Installation,” you might want to move the location of the sample WSA Web application that is installed with OpenEdge to avoid any dependency on the OpenEdge installation. You also might want to specify a separate Web service deployment directory for a given WSA instance for the same reason, especially if you continue to use the sample Web application in its initial OpenEdge installation location. Otherwise, if you upgrade or re-install OpenEdge and do not manually backup them up, you can lose your current deployed Web services during the upgrade.

You can change the deployment directory for all Web services deployed to a given WSA instance by modifying the Web application descriptor file (web.xml). For a description of this file, see the information on the WSA as a Web application in Chapter 5, “Managing the Web Services Adapter.” This file is initially installed at the following OpenEdge installation location for the WSA sample Web application:

```
OpenEdge-Install-Directory/servlets/wsa/WEB-INF/web.xml
```

**Note:** These instructions assume that you have not changed the sample WSA Web application directory from its installed location.

**To specify a new deployment location for the Web services of a WSA instance:**

1. Open the `web.xml` that defines the WSA servlet instance for editing in a text editor.
2. Locate the following XML in the file by searching for "deploymentDir" within the servlet definition for your WSA instance. For example:

```
<servlet>
    <servlet-name>wsa1_servlet</servlet-name>
    <display-name>Web Services Adapter servlet 1</display-name>
    <!-- Enter an optional description of the Web Services Adapter servlet and uncomment this element if supported by the JSE -->
    <description>Web Services Adapter servlet 1</description>
    -->
    . . .
    <!-- To change the deployment directory from the default, uncomment this parameter and set the value to the correct directory. -->
    <init-param>
        <param-name>deploymentDir</param-name>
        <param-value>C:/work/deployment/</param-value>
    </init-param>
    -->
    . . .
</servlet>
```

3. Remove the XML comment tags around the `<init-param>` element and specify the new deployment directory as the `<param-value>` element value (C:/work/deployment/).
4. Save and close the file with the new information.

5. If you have any existing deployed Web services for this WSA instance, move the existing WSA instance directory to the new deployment directory. For example, if the WSA instance directory is `wsa1`, move the entire directory subtree to the new deployment directory as follows, using the appropriate operating system commands:

```
OpenEdge-Install-Directory/servlets/wsa/wsaw/* → C:/work/deployment/wsaw/*
```

6. Restart the JSE or Web server.

You can now deploy and otherwise manage Web services for the WSA instance, `wsa1`. 
Enabling a Web service

You can enable a Web service using OpenEdge Explorer or \texttt{WSAMAN} utility’s \texttt{enable} function. This section describes how to enable a Web service using OpenEdge Explorer. For more information on using the \texttt{WSAMAN} utility to enable a Web service, see Chapter 8, “Using the WSA Management Utility (WSAMAN)” and Appendix B, “Command and Utility Reference.”

See \textit{OpenEdge Management and OpenEdge Explorer: Getting Started} for complete information on enabling a Web service using OpenEdge Explorer. You can also access this information from the online help.
Administering a deployed Web service

Once a Web service is deployed and enabled, you can administer it by performing the actions described in Table 6–5.

Table 6–5: Actions for administering a deployed and enabled Web service

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable and enable it</td>
<td>Makes the Web service unavailable to Web service clients; makes it available again</td>
</tr>
<tr>
<td>Export and import it</td>
<td>Creates a backup file of the deployed Web service; uses the information in the backup file to deploy a Web service</td>
</tr>
<tr>
<td>Note: Export creates a Web Service Definition (WSD) file; import requires an existing WSD file.</td>
<td></td>
</tr>
<tr>
<td>Get and set its properties</td>
<td>Provides the value of its properties; resets one of its properties</td>
</tr>
<tr>
<td>Get and reset its statistics</td>
<td>Displays the value of each Web service statistic; reinitializes the statistics</td>
</tr>
<tr>
<td>Query its deployment information and display its status (enabled or disabled)</td>
<td>Provides the Web service’s status (enabled or disabled), friendly name, namespace, SOAP action, and SOAP message style</td>
</tr>
<tr>
<td>Update it</td>
<td>Lets you change the Web service’s WSM file without undeploying and redeploying it</td>
</tr>
<tr>
<td>Undeploy it</td>
<td>Removes the files created when the Web service was deployed</td>
</tr>
</tbody>
</table>

To perform these functions, you can use OpenEdge Explorer or the WSA MAN utility. For more information, see the online help, Chapter 8, “Using the WSA Management Utility (WSAMAN),” and Appendix B, “Command and Utility Reference.”
Web service file management for deployed Web services

Deployed Web services involve several files. Among them are WSDL files (used by clients at development time) and WSD files (produced when a Web service is exported).

Options for providing the WSDL file to a client

There are several ways to provide the WSDL file to the client, including both WSA-supported options and alternative options. For more information, see the sections on Web service clients accessing OpenEdge Web services in OpenEdge Development: Web Services.

Exporting and importing Web services

When a Web service is exported, the WSD file produced contains all the information needed to import the exported Web service. When an exported Web service is imported, a WSD file must be supplied to provide the information required to re-create the Web service.

Notes: The WSD file contains a complete copy of the Web service properties of the original Web service. When an import is performed, the imported Web service has the same properties as the original.

An import requires the WSD file be accessible to the local machine. An export produces the WSD file accessible to the local machine—that is, the machine where OpenEdge Explorer or the WSAMAN utility run.

For more information exporting and importing Web services using OpenEdge Explorer, see the online help.

For more information on exporting and importing Web services using the WSAMAN utility, see Chapter 8, “Using the WSA Management Utility (WSAMAN)” and Appendix B, “Command and Utility Reference.”
Monitoring and tuning Web services

To monitor a Web service, inspect its statistics. To tune it, adjust its properties. You can perform these tasks using the OpenEdge Explorer or the WSAMAN utility. Table 6–6 lists the types of statistics gathered for a Web service.

Table 6–6: Statistics for a Web service

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Requests</td>
<td>Total requests handled by the Web service</td>
</tr>
<tr>
<td>Number of Active Requests</td>
<td>Requests currently being serviced</td>
</tr>
<tr>
<td>Number of SOAP Faults</td>
<td>Error returns by the Web service application</td>
</tr>
<tr>
<td>Number of Objects Not Found</td>
<td>Calls to unidentified Web service methods</td>
</tr>
<tr>
<td>Number of Errors</td>
<td>Total errors returned for requests to the Web service, with errors broken down for each of several error categories after the first error is reported</td>
</tr>
<tr>
<td>Number of AppServer Connections</td>
<td>Total AppServer connections being managed by this Web service</td>
</tr>
<tr>
<td>Number of Proxy AppObjects</td>
<td>Total AppObjects registered for all clients of this Web service</td>
</tr>
<tr>
<td>Number of Proxy SubAppObjects</td>
<td>Total SubAppObjects registered for all clients of this Web service</td>
</tr>
<tr>
<td>Number of Proxy ProcObjects</td>
<td>Total ProcObjects registered for all clients of this Web service</td>
</tr>
</tbody>
</table>

For more information on the statistics of a Web service, see OpenEdge Explorer online help.

If the Web service uses a NameServer and is session free, the Web service has a connection pool, with properties you can adjust. If the Web service is session managed, it can be set to store multiple requests to the service or repeat them. In either case, for more information, see Appendix A, “Reference to OpenEdge Web Service Properties” and the OpenEdge Explorer online help.
Typical Web service administration scenarios

This section contains some typical Web service administration scenarios.

Deploying a new version of your Web service

In this scenario, you deploy a Web service, then the developer hands you a new WSM file generated by ProxyGen. You want to deploy a new version of the Web service using the new WSM file. The technique you use depends on which of the following the environments the Web service runs in:

- Production environment
- Development environment

To deploy a new version of a Web service running in a production environment:

1. Deploy the new version of the Web service with a new friendly name, and new target namespace.
2. Enable the new version of the Web service.
3. Notify clients that the new version of the Web service is available, and instruct them to update their clients based on the new WSDL.
4. Once there are no clients using the old version of the Web service, disable it.
5. Export the Web service, to create a backup. This provides a WSD file of the original version.
6. Remove the old version of the Web service.

**Note:** In a production environment, Progress Software Corporation recommends that you keep the older and the newer versions running simultaneously until all clients of the Web service have converted to the newer version. Only then, remove the older version.

To deploy a new version of a Web service running in a development environment:

1. Disable the Web service.
2. Export the Web service. This provides a WSD file of the original version, to create a backup.
3. Update the Web service with the new WSM file.
4. Enable the Web service.

**Caution:** Progress Software Corporation recommends you never use the Update function in a production environment.
Changing Web service properties

In this scenario, you deploy a Web service, then you want to change a property that takes effect only when the Web service is disabled and re-enabled.

**To change Web service properties:**

1. Notify all clients that the Web service is going to be disabled at a certain time.
2. Wait until there are no clients are using the Web service.
3. Disable the Web service.
4. Modify the Web service’s properties, as desired, using OpenEdge Explorer or the **WSAMAN** utility’s `setprops` function.
5. Enable the Web service.
6. Notify clients that the Web service is available.

You do not have to disable and enable a Web service to modify its `serviceLoggingLevel` property, which changes the level of information provided in log messages, or its `serviceFaultLevel` property, which changes the level of information provided in SOAP Fault response messages. For more information, see the information on debugging OpenEdge Web services in *OpenEdge Development: Web Services*. 
Web Services Adapter Security Configurations

This chapter provides an overview of the security provided by the Web Services Adapter (WSA), describes its initial settings, and provides an alphabetical list of configuration instructions for a variety of situations, as described in the following sections:

- Overview
- Initial settings
- Alphabetical list of configuration instructions
Overview

When WSA security is enabled, the JSE and WSA work in tandem to authenticate the Web service client and to check that the client has the privileges to execute the requested operation.

Role of the JSE

The WSA’s `web.xml` file can be set to tell the JSE that one or more of its URL paths require user authentication. When this is set, the JSE intercepts each HTTP request destined for that URL. For example, `web.xml` can be set so that users who try to access a WSA instance’s Web service application URL are authenticated and required to have the role `WsaWebServicesUser`. In fact, this is how security can be initialized during OpenEdge installation.

Note: The entries in `web.xml` are case-sensitive.

For example, when the JSE intercepts an HTTP request destined for the WSA instance’s Web service application URL:

1. The JSE performs HTTP Basic authentication using the user’s ID and password.
2. If the user ID and password are valid, the JSE checks that the user ID has been granted access to the Role `WsaWebServicesUser`.
3. If the user ID has been granted access, the JSE passes the HTTP request and the authenticated user ID to the WSA.

Role of the WSA

The WSA’s `ubroker.properties` file can be set to also require the WSA to verify that the JSE performs user authentication for particular URL paths.

Note: The entries in `ubroker.properties` are case-sensitive.

For example, when the WSA gets an HTTP request from the JSE:

1. The WSA first checks its `ubroker.properties` values to determine whether the URL requires the JSE to perform user authentication.
2. If it does, the WSA verifies that the JSE has passed it a valid user ID—just in case the JSE configuration becomes corrupted.
3. When it verifies that it has a valid user ID, the WSA checks whether the URL is for a WSA administrative function.

   If the URL is for a WSA administrative function, then:

   a. The WSA determines which Role the user is in and gives the user the security privileges associated with that Role.

   b. The WSA determines whether the user’s privileges allow it to execute the administrative function. If yes, the administrative function is executed.

   If the URL is not for a WSA administrative function, the WSA processes the HTTP/SOAP request as a web service operation.

JSE and WSA security files

JSE security and WSA security involve different files. JSE security depends primarily on the settings of the WSA’s web.xml file, while WSA security depends primarily on the settings of the WSA’s ubroker.properties file.

When the WSTK installation runs, it tailors the web.xml and ubroker.properties files, initializing JSE and WSA security as specified during installation. You can use these initial settings or modify them as desired.

Caution: The installation tailors the web.xml file defined in OpenEdge-Install-Directory/servlets/wsa. If you want to enable security in another web.xml, use a JSE that creates its own WSA-web.xml file, you must manually edit it, inserting the WSA Web application defined in OpenEdge-Install-Directory/servlets/wsa.

Security features available

The JSE provides user authentication and role authorization. To these security features, the WSA adds the following, all of which are optional:

- Requiring the JSE to perform user authentication and role authorization for any combination of the following optional features:
  - Accessing WSDL files
  - Running Web service applications
  - Performing WSA administration
- Customizing, per-role, administration privileges for WSA administrators
- Specifying additional user-defined administration roles
- Customizing default-administrator-role privileges
Disabling the WSA from responding to all user requests from any combination of the following:

- Accessing Web service applications
- Accessing WSDL files and/or WSDL file catalogs
- Performing WSA administration
Initial settings

JSE security is initially set in OpenEdge-Install-Directory/servlets/wsa/web.xml as shown in Table 7–1.

Table 7–1: Initial settings for JSE security

<table>
<thead>
<tr>
<th>Security feature</th>
<th>Status</th>
<th>web.xml setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandatory authentication and authorization for users of Web service applications</td>
<td>Disabled</td>
<td>The security-context for Web service applications, wsa-root-url, is commented out.</td>
</tr>
<tr>
<td>Mandatory authentication and authorization for users of WSDL files</td>
<td>Disabled</td>
<td>The security-context for WSDL file access, wsa-root-url/wsd1, is commented out.</td>
</tr>
<tr>
<td>Mandatory authentication and authorization for WSA administration users with either the PSCAdmin or PSCOper role</td>
<td>Enabled</td>
<td>The security-context for administration access, wsa-root-url/admin, is not commented out.</td>
</tr>
</tbody>
</table>

WSA security is initially set in ubroker.properties, as shown in Table 7–2.

Table 7–2: Initial settings for WSA security

<table>
<thead>
<tr>
<th>Security feature</th>
<th>Status</th>
<th>ubroker.properties setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access to administration operations</td>
<td>Enabled</td>
<td>adminEnabled=1</td>
</tr>
<tr>
<td>Access to Web service applications</td>
<td>Disabled</td>
<td>webAppEnabled=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> Before any web service request can be accepted, this property must be set on (1).</td>
</tr>
<tr>
<td>Access to WSDL files</td>
<td>Disabled</td>
<td>enableWsdl=0, enableWsdlListings=0</td>
</tr>
<tr>
<td>Mandatory authentication and authorization of users with an administrator role</td>
<td>Enabled</td>
<td>adminAuth=1</td>
</tr>
<tr>
<td>Mandatory authentication and authorization of users of Web service applications</td>
<td>Disabled</td>
<td>appAuth=0</td>
</tr>
<tr>
<td>Definition of administrator roles</td>
<td>PSCAdmin, PSCOper</td>
<td>adminRoles=PSCAdmin, PSCOper</td>
</tr>
</tbody>
</table>

**Note:** A developer might develop and test a Web service using a nonsecure WSA, then (if necessary) either deploy it to a secure WSA or secure the test WSA.
Alphabetical list of configuration instructions

Working with WSA security involves working with the web.xml file and with settings in the ubroker.properties file.

Disabling and enabling security in the web.xml (OpenEdge-Install-Directory/servlets/wsa) file involves commenting and uncommenting security-constraints.

In XML, comments start with the start-comment (<!--) and end with the end-comment (--> tag.

To comment out an XML security-constraint, place an XML start-comment before the XML element tag that begins the entry and place an XML end-comment after the XML tag that terminates the <security-constraint> element, as in the following example:

```
<!--
   <security-constraint>
     ...
   </security-constraint>
-->
```

The web.xml file included with the sample Web service application includes embedded comments with additional information on changes you might need to make.

The configuration instructions are contained in the following sections:

- Controlling access to Web services, WSDL, and WSA administration using one JSE security-constraint
- Controlling access to Web services, WSDL, and WSA administration using user-authorization role-names
- Customizing the default administrator’s permissions and action settings
- Customizing the PSCOper role’s permissions and action settings
- Customizing WSA administration roles
- Disabling access to all Web services, to all WSDL, or to all WSA administration
- Disabling authentication for WSA administration
- Disabling the deploying and undeploying of Web services
- Disabling WSDL listings
- Enabling authentication for Web services
- Enabling authentication for WSDL
- Enabling multiple user roles to access Web services, WSDL, or WSA administration
- Enabling Web services per user and per application
Controlling access to Web services, WSDL, and WSA administration using one JSE security-constraint

The WSA provides three different JSE security-constraints (residing in the WSA’s web.xml file) to control access to the following:

- Web service applications
- WSDL
- WSA administration

Controlling access to all three with a single constraint is not supported.

To control access, in the WSA instance’s web.xml file, uncomment or create a security-constraint for accessing each function in your desired combination, as shown in Table 7–3.

Table 7–3: Controlling Web service, WSDL, and administration access using JSE security constraints

<table>
<thead>
<tr>
<th>For accessing . . .</th>
<th>The security-constraint might look like this . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web services</td>
<td>&lt;security-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;url-pattern&gt;/wsa1/&lt;/url-pattern&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/security-constraint&gt;</td>
</tr>
<tr>
<td>WSDL</td>
<td>&lt;security-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;url-pattern&gt;/wsa1/wsdl/*&lt;/url-pattern&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/security-constraint&gt;</td>
</tr>
<tr>
<td>WSA administration</td>
<td>&lt;security-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;url-pattern&gt;/wsa1/admin/*&lt;/url-pattern&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/security-constraint&gt;</td>
</tr>
</tbody>
</table>
Controlling access to Web services, WSDL, and WSA administration using user-authorization role-names

You can control access to all of a WSA instance’s Web services, WSDL, and WSA administration, or to any desired combination of these functions, using user-authorization role-names.

To control access to Web services, WSDL, and WSA administration:

1. Choose user-authorization role-names to identify users who can access your desired combination of functions.

2. In each security-constraint, uncommented or created in the previous step, add a role-name element for each user-authorization role-name you chose. The syntax of a role-name element is:

   \[
   \text{<auth-constraint>}
   \text{<role-name>name</role-name>}
   \text{</auth-constraint>}
   \]

For example, if you want to grant access to all of a WSA’s Web services, WSDL, and WSA administration and grant access to the user-authorization role-names \text{PSCAdmin} and \text{GuestAdmin}, you might modify the security-constraints as shown in Table 7–4.

Table 7–4: Controlling Web service, WSDL, and administration access using role names

<table>
<thead>
<tr>
<th>For accessing . . .</th>
<th>The modified security-constraint might look like this . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web services</td>
<td>&lt;security-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;url-pattern&gt;/wsa1/&lt;/url-pattern&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;auth-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;role-name&gt;PSCAdmin&lt;/role-name&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;role-name&gt;GuestAdmin&lt;/role-name&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/auth-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/security-constraint&gt;</td>
</tr>
<tr>
<td>WSDL</td>
<td>&lt;security-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;url-pattern&gt;/wsa1/wsdl/*&lt;/url-pattern&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;auth-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;role-name&gt;PSCAdmin&lt;/role-name&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;role-name&gt;GuestAdmin&lt;/role-name&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/auth-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/security-constraint&gt;</td>
</tr>
</tbody>
</table>
3. Modify the properties of the WSA instance to require JSE authentication of all users of your desired combination of functions. The technique for doing so depends on whether the WSA instance is local (residing on the AdminServer machine) or remote (not residing on the AdminServer machine).

If the WSA instance is local, using OpenEdge Explorer, select the WSA instance, select Configuration, and select the Security tab.

Then, click Edit and enable the checkboxes to require JSE user authentication of all users in your desired combination of functions, as follows:

- Require WSA administration authorization
- Require Web Services authorization
- Require WSDL retrieval authorization

For more info, see the online help.

If the WSA instance is remote, edit the ubroker.properties file for the WSA. In the section for the WSA instance, for each function in your desired combination, set the property that enables JSE authentication of all users of that function, as shown in Table 7–5.

Table 7–4: Controlling Web service, WSDL, and administration access using role names

<table>
<thead>
<tr>
<th>For accessing . . .</th>
<th>The modified security-constraint might look like this . . .</th>
</tr>
</thead>
</table>
| WSA administration   | <security-constraint>
|                      |   <web-resource-collection>
|                      |      <url-pattern>/wsa1/admin/*</url-pattern/>*
|                      |   <auth-constraint>
|                      |      <role-name>PSCAdmin</role-name>
|                      |      <role-name>GuestAdmin</role-name>
|                      |   </auth-constraint>
|                      | </web-resource-collection>
|                      | </security-constraint>                                   |

Table 7–5: Requiring Web service user authorization for Web service, WSDL, and administration access

<table>
<thead>
<tr>
<th>To require JSE user authentication for all users of . . .</th>
<th>Set this property . . .</th>
<th>To . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web services</td>
<td>appAuth</td>
<td>1</td>
</tr>
<tr>
<td>WSDL</td>
<td>wsdlAuth</td>
<td>1</td>
</tr>
<tr>
<td>WSA administration</td>
<td>adminAuth</td>
<td>1</td>
</tr>
</tbody>
</table>

For more information on the properties of a WSA instance, see the comments in the ubroker.properties file.
Customizing the default administrator’s permissions and action settings

When you install the WSA, you can choose to enable security. In this case, security is preset so that any user of WSA administration authorized by the JSE whose role is not listed in the WSA’s list of administration roles is assigned the permissions defined for the default administrator.

To change the permissions and action settings for the default administrator:

1. Edit the ubroker.properties file using a text editor.
2. Locate the [AdminRole] group.
3. Change the individual permission properties to have the read, write, and delete actions you choose. For example:

   ```
   [AdminRole]
   apps_defaults=
   apps_enable=
   apps_props=
   apps_stats=
   servlet_props=
   servlet_services=read
   servlet_stats=
   ```

Customizing the PSCOper role’s permissions and action settings

When you install the WSA, you can choose to enable security. In this case, security is preset so that a WSA instance performs per-role access checking to individual administration functions when the adminRoles property is not set to blank. You can customize any role defined in the adminRoles property.

To customize the PSCOper role:

1. Edit the ubroker.properties file using a text editor.
2. Locate the [AdminRole.PSCOper] group.
3. Change the individual permission properties to have the read, write, and delete actions you choose. For example:

   ```
   [AdminRole.PSCOper]
   apps_defaults=read
   apps_enable=read
   apps_props=read
   apps_stats=read
   servlet_props=read
   servlet_services=read
   servlet_stats=read
   ```
Customizing WSA administration roles

When you install the WSA, you can choose to enable security. In this case, security is preset to enable access to WSA administration with two predefined administrator roles, PSCAdmin (for system administrators) and PSCOper (for system operators). However, you can define your own custom administrator roles.

To define your own custom administrator roles:

1. Enable the JSE to authenticate users and grant them access if they hold a membership in one of your customized administrator roles. To do so, in the WSA’s web.xml file, in the security-constraint for WSA administration, add a role-name element for each of your customized administrator roles.

A security-constraint for WSA administration looks like the following:

```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/wsa1/admin/*</url-pattern/>
  </web-resource-collection>
  <auth-constraint>
    <role-name>tempadmin1</role-name>
    <role-name>tempadmin2</role-name>
  </auth-constraint>
</security-constraint>
```

The syntax for a role-name element is:

**Syntax**

```
<auth-constraint>
  <role-name>name</role-name>
</auth-constraint>
```

For example, if you created the new administrator roles tempadmin1 and tempadmin2, you might modify the security-constraint for WSA administration to appear as follows:

```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/wsa1/admin/*</url-pattern/>
    <auth-constraint>
      <role-name>tempadmin1</role-name>
      <role-name>tempadmin2</role-name>
    </auth-constraint>
  </web-resource-collection>
</security-constraint>
```

2. Add an administrator role to the WSA security. Using a text editor, edit the ubroker.properties file from which the WSA is initialized, as follows:

   a. Locate the [AdminRole.PSCAdmin] group.

      Each administrator role definition resides in groups that begin with “[AdminRole.”, followed by the role-name, and ending with “]”.

   b. Copy the group once for each new administrator role you want to create.

   c. Choose a new role-name. Change the group’s role-name to reflect the new role name.
d. Edit the group’s Permission properties and actions as desired. For example:

```
[AdminRole.PSCAdmin]
  apps_defaults=read,write,
  apps_enable=read,write,
  apps_props=read,write,
  apps_stats=read,write,
  servlet_props=read,write,
  servlet_services=read,write,
  servlet_stats=read,write
```

3. Modify the `adminRole` property of the WSA to add the new role-names to the list of role-names. The technique for doing so depends on whether the WSA instance is local (residing on the AdminServer machine) or remote (not residing on the AdminServer machine).

If the WSA is local, using OpenEdge Explorer, select the WSA instance. Then, in the WSA’s Security panel, in the list of Admin roles, add the new role-names.

If the WSA instance is remote, using a text editor, edit the `ubroker.properties` file. In the `adminRoles` property, add the new role-names to the comma-separated list of existing role names, as shown:

```
[WSA]
  .
  .
  .
  adminRoles=PSCAdmin,PSCOper
```

**Disabling access to all Web services, to all WSDL, or to all WSA administration**

Given a WSA instance, you can disable access to all of its Web services, to all of its WSDL, or to all of its WSA administration. The technique for doing so depends on whether the WSA instance resides locally (on the AdminServer machine) or remotely (not on the AdminServer machine).

If the WSA instance resides locally, using OpenEdge Explorer, click the WSA instance and click **Configuration**. Select **Security** to bring up the Security properties. Then, select the fields on the Security panel to disable access to all Web services, all WSDL, or all WSA administration, as desired. For more information on accessing the Security properties, see the “Controlling access to Web services, WSDL, and WSA administration using user-authorization role-names” section on page 7–8.
If the WSA instance is remote, using a text editor, edit the `ubroker.properties` file. In the section for the WSA instance, for each function that you want to disable, set the corresponding property, shown in Table 7–6.

### Table 7–6: Disabling access to Web services, WSDL, and administration

<table>
<thead>
<tr>
<th>To disable access to . . .</th>
<th>Set this property . . .</th>
<th>To . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web services</td>
<td><code>webAppEnabled</code></td>
<td>0</td>
</tr>
<tr>
<td>WSDL</td>
<td><code>enableWsdl</code></td>
<td>0</td>
</tr>
<tr>
<td>WSA administration</td>
<td><code>adminEnabled</code></td>
<td>0</td>
</tr>
</tbody>
</table>

For more information on the properties of a WSA instance, see the comments in the `ubroker.properties` file.

**Note:** If you disable all WSA administration access while the WSA instance is running, you must stop and restart the JSE to re-enable WSA administration access.

### Disabling authentication for WSA administration

When you install the WSA, you can choose to enable WSA security to perform authentication for WSA administration. This authentication can be disabled.

#### To disable authentication for WSA administration:

1. Using a text editor, edit the WSA instance’s `web.xml` file, commenting out the security-constraint holding the URL path of the WSA administrator, as shown:

   ```xml
   <!--
   <security-constraint>
     <web-resource-collection>
       <url-pattern>/wsa1/admin/*</url-pattern>
     </web-resource-collection>
   </security-constraint>
   -->
   ``

   **Note:** In the actual `web.xml` file, the URL path of the WSA administrator is in bold.

2. Using OpenEdge Explorer (or the WSAMAN utility), disable (or set to 0) the WSA instance’s `adminAuth` property:

   - If the WSA instance is local (the WSA resides on the AdminServer machine), using OpenEdge Explorer, select the WSA instance, right-click, and choose **Properties**. Select Security to display the Security panel. In the Security panel, disable authentication for WSA administration.
• If the WSA instance is remote (the WSA instance does not reside on the AdminServer machine), using a text editor, edit the ubroker.properties file. In the properties for the WSA instance, set adminAuth to 0 (zero), as shown:

```
[WSA]
  
  adminAuth=1
```

Disabling the deploying and undeploying of Web services

Given a WSA instance, WSA lets you disable the deploying and undeploying of Web service applications without disabling other functions.

To disable the deploying and undeploying of Web services:

1. If the WSA is local, configure the WSA to not use the “all administrators have all privileges” rule. To do this:
   a. Use OpenEdge Explorer, select the WSA instance, click Configuration, and select Security.
   b. Change the WSA instance's list of administrator roles by selecting at least one of the listed administrator roles.

   When no administrator roles are selected, the WSA invokes the “all administrators have all privileges” rule. When at least one role is selected, the WSA enforces per-administrator-role Permissions.

   Normally, the WSA administration URL’s <security-constraint> setting is configured to allow both the PSCAdmin and PSCOper roles. Selecting these predefined roles is the easiest way to manage WSA administration.

2. If the WSA instance is remote, using a text editor, edit the ubroker.properties file. In either or both of the [AdminRole.PSCAdmin] and [AdminRole.PSCOper] groups, as desired, edit the servlet_services property from read, write, delete to read.

   For more information on the WSA properties in ubroker.properties, see the documentation comments in ubroker.properties.

Disabling WSDL listings

Each WSA instance provides a listing of the WSDL files associated with the Web service applications deployed to the WSA instance. For each WSDL file available, the listing provides a URL for accessing it.

When you install the WSA, you can choose to enable security. In this case, security is preset to enable access to WSDL listings. But you can disable this access. The technique for doing so depends on whether the WSA instance resides locally (on the AdminServer machine) or remotely (not on the AdminServer machine).
Note: Neither technique disables access to the WSA instance’s WSDL files individually.

If the WSA instance resides locally, using OpenEdge Explorer, select the WSA instance, click Configuration, and select Security to bring up the Security properties. In the Security tab, disable WSDL listings for the WSA instance.

If the WSA instance resides remotely, in the ubroker.properties file from which the WSA instance’s properties are initialized, in the section for the WSA instance, set the enableWsdllisting property to 0, as shown:

```
[WSA]
enableWsdllistings=1
```

Enabling authentication for Web services

When you install the WSA, you can choose to enable security. In this case, WSA security is preset so that users who try to access Web service applications are not authenticated. This authentication can be enabled.

To enable authentication for Web services:

1. Choose an authorization role name to identify users who get access to all of a WSA instance’s Web service applications.

2. Using a text editor, edit the WSA instance’s web.xml file as follows:
   a. Uncomment or add a `<security-constraint>` element for the URL path for Web service applications. The security-constraint might appear as follows:

```
<!--
<security-constraint>
  <web-resource-collection>
    <url-pattern>/wsa1/</url-pattern>
  </web-resource-collection>
  <auth-constraint />
</security-constraint>
-->
```

b. Add a `<role-name>` element to the `<security-constraint>` element, to hold the authorization role name you chose. A `<role-name>` element has the following syntax:

```
<auth-constraint>
  <role-name>name</role-name>
</auth-constraint>
```
For example, after you add the role names webservicesrole1 and webservicesrole2, the `<security-constraint>` element might appear as follows:

```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/wsa1/</url-pattern>
    <auth-constraint>
      <role-name>webservicesrole1</role-name>
      <role-name>webservicesrole2</role-name>
    </auth-constraint>
  </web-resource-collection>
</security-constraint>
```

3. Modify the JSE to add your user’s accounts and grant them membership to the chosen role.

4. Modify the WSA instance’s properties to enable authorization for Web services. Use one of the following techniques:

   - If the WSA instance is local (the WSA resides on the AdminServer machine), using OpenEdge Explorer, select the WSA instance, click Configuration, and select Security to bring up the Security tab. In the WSA instance’s Security tab, enable authentication for Web services.
   
   - If the WSA instance is remote (the WSA instance does not reside on the AdminServer machine), use a text editor.
   
   - In the `ubroker.properties` file, in the properties for the WSA instance, set `appAuth` to 1 (one). The possible values for `appAuth` are listed in Table 7–7.

### Table 7–7: Setting the appAuth property

<table>
<thead>
<tr>
<th>Set appAuth to . . .</th>
<th>To . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (one)</td>
<td>Enable authentication for users of Web services</td>
</tr>
<tr>
<td>0 (zero)</td>
<td>Disable authentication for users of Web services</td>
</tr>
</tbody>
</table>

For example:

```
[wSA]
  .
  .
  .
  appAuth=0
```
Enabling authentication for WSDL

When you install the WSA component of the WSTK, you can choose to enable security. In this case, WSA security is preset so that users who try to access WSDL are not authenticated. This authentication can be enabled.

To enable authentication for WSDL:

1. Choose an authorization role name to identify users who get access to all of a WSA instance’s WSDL.

2. Using a text editor, edit the WSA instance’s web.xml file as follows:

   a. Uncomment or add a `<security-constraint>` element for the URL path for WSDL. The `<security-constraint>` element might appear as follows:

   ```xml
   <security-constraint>
     <web-resource-collection>
       <url-pattern>/wsa1/wsdl/*</url-pattern>
     </web-resource-collection>
     <auth-constraint>
       <role-name>wsdlrole1</role-name>
       <role-name>wsdlrole2</role-name>
     </auth-constraint>
   </security-constraint>
   ```

   b. Add a `<role-name>` element to the `<security-constraint>` element to hold the authorization role name you chose. A `<role-name>` element has the following syntax:

   ```xml
   <role-name>name</role-name>
   ```

   For example, after you add the role-names `wsdlrole1` and `wsdlrole2`, the `<security-constraint>` element might appear as follows:

   ```xml
   <security-constraint>
     <web-resource-collection>
       <url-pattern>/wsa1/</url-pattern>
       <auth-constraint>
         <role-name>wsdlrole1</role-name>
         <role-name>wsdlrole2</role-name>
       </auth-constraint>
     </web-resource-collection>
   </security-constraint>
   ```

3. Modify the JSE to add your user’s accounts and grant them membership to the chosen role.

4. Modify the WSA instance’s properties to enable authorization for WSDL. Use one of the following techniques:

   - If the WSA instance is local (the WSA resides on the AdminServer machine), using OpenEdge Explorer, select the WSA instance, click Configuration, and choose Properties. Select Security to display the Security tab. In the Security tab, enable authentication for WSDL.
If the WSA instance is remote (the WSA instance does not reside on the AdminServer machine), using a text editor, in the ubroker.properties file from which the WSA instance is initialized, in the properties for the WSA instance, set wsdlAuth to 1 (one). The possible values for wsdlAuth are listed in Table 7–8.

Table 7–8: Setting the wsdlAuth property

<table>
<thead>
<tr>
<th>Set wsdlAuth to . . .</th>
<th>To . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (one)</td>
<td>Enable authentication for users of WSDL</td>
</tr>
<tr>
<td>0 (zero)</td>
<td>Disable authentication for users of WSDL</td>
</tr>
</tbody>
</table>

For example:

```
[WSA]
.
.
.wsdlAuth=0
```

Enabling multiple user roles to access Web services, WSDL, or WSA administration

You can enable multiple roles to access Web service applications, WSDL files, WSA administration, or any combination of the three.

To enable multiple roles, use a text editor to edit the WSA instance’s web.xml file.

To enable multiple user roles:

1. Find the <security-constraint> element for accessing Web services, WSDL, or WSA administration, as shown in Table 7–9.

Table 7–9: Enabling multiple user roles for Web services, WSDL, and administration

<table>
<thead>
<tr>
<th>For accessing . . .</th>
<th>The security-constraint might look like this . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web services</td>
<td>&lt;security-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;url-pattern&gt;/wsa1/&lt;/url-pattern&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/security-constraint&gt;</td>
</tr>
</tbody>
</table>
Table 7–9: Enabling multiple user roles for Web services, WSDL, and administration (2 of 2)

<table>
<thead>
<tr>
<th>For accessing . . .</th>
<th>The security-constraint might look like this . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSDL</td>
<td>&lt;security-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;url-pattern&gt;/wsa1/wsdl/*&lt;/url-pattern&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/security-constraint&gt;</td>
</tr>
<tr>
<td>WSA administration</td>
<td>&lt;security-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;url-pattern&gt;/wsa1/admin/<em>&lt;/url-pattern/</em>&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/security-constraint&gt;</td>
</tr>
</tbody>
</table>

2. Add the desired number of <role-name> elements to the <security-constraint> element, where each element contains exactly one role name.

A <role-name> element has the following syntax:

**Syntax**

```
<auth-constraint>
  <role-name>name</role-name>
</auth-constraint>
```

For example, you can set up the role names WSUser1 and WSUser2 for accessing Web services, WSDUser1 and WSDUser2 for accessing WSDL, and WSAdminUser1 and WSAdminUser2 for accessing WSA administration, as shown in Table 7–10.

Table 7–10: Setting security constraints for multiple user roles on Web services, WSDL, and administration (1 of 2)

<table>
<thead>
<tr>
<th>For accessing . . .</th>
<th>The resulting security-constraint might look like this . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web services</td>
<td>&lt;security-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;url-pattern&gt;/wsa1/&lt;/url-pattern&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;auth-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;role-name&gt;WSUser1&lt;/role-name&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;role-name&gt;WSUser2&lt;/role-name&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/auth-constraint&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/web-resource-collection&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/security-constraint&gt;</td>
</tr>
</tbody>
</table>
Enabling Web services per user and per application

When you install the WSA, you can choose to enable security. In this case, security is preset to allow any user authenticated and authorized by the JSE to access to any deployed Web service application. You can, however, restrict access to Web service applications per user and per application. The technique involves creating a WSA instance for each unique user community, setting the security of each WSA instance to restrict access to a particular user community, and deploying to each WSA instance Web services only for its user community.

For example, suppose the Acme Company wants to deploy the applications described in Table 7–11 as Web services.

Table 7–11: Enabling Web services per user and per application

<table>
<thead>
<tr>
<th>This application . . .</th>
<th>Available to . . .</th>
<th>Lets its users . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting (Query Only)</td>
<td>All department managers and their superiors</td>
<td>Read any record in the Accounting system</td>
</tr>
<tr>
<td>Accounting (Update)</td>
<td>Members of the Accounting department only</td>
<td>Read, modify, and create records in the Accounting system</td>
</tr>
<tr>
<td>Human Resources</td>
<td>Members of the Human Resources department only</td>
<td>Read, modify, and create records in the Human Resources system</td>
</tr>
<tr>
<td>Work Tracking</td>
<td>All employees</td>
<td>Record the time spent on each project</td>
</tr>
</tbody>
</table>
To enable Web services per user and per application:

1. Set up the WSA instances, security-constraint URL-patterns, and role names as shown in Table 7–12.

Table 7–12: Sample Web service security constraints by application and role name

<table>
<thead>
<tr>
<th>Application</th>
<th>WSA instance name</th>
<th>Security-constraint URL-pattern</th>
<th>Role-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting-Full</td>
<td>acctg</td>
<td>/acctg/</td>
<td>Acctdept</td>
</tr>
<tr>
<td>Accounting-Query</td>
<td>acqntgquery</td>
<td>/acctgquery/</td>
<td>DeptMgr</td>
</tr>
<tr>
<td>Human Resources</td>
<td>Hr</td>
<td>/hr/</td>
<td>Hrdept</td>
</tr>
<tr>
<td>Work-Tracking</td>
<td>worktracking</td>
<td>/worktracking/</td>
<td>Users</td>
</tr>
</tbody>
</table>

2. In the properties of each WSA instance, require user authorization to Web service applications. For example:

   - If the WSA is local, using OpenEdge Explorer, select the WSA instance, right-click, and select Properties. Select Security to display the Security panel. In the Security panel, add user authorization to Web services.
   - If the WSA instance is remote, using a text editor, edit the ubroker.properties file. Add user authorization to Web services.

3. In the JSE’s user database, assign each employee the appropriate role, as shown in Table 7–13.

Table 7–13: Sample Web service user roles

<table>
<thead>
<tr>
<th>To these employees . . .</th>
<th>Assign this role . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>All department managers and their superiors.</td>
<td>DeptMgr</td>
</tr>
<tr>
<td>All employees of the company.</td>
<td>Users</td>
</tr>
<tr>
<td>All members of the Accounting department.</td>
<td>Acctdept</td>
</tr>
<tr>
<td>All members of the Human Resources department.</td>
<td>Hrdept</td>
</tr>
</tbody>
</table>
The resulting web.xml file follows:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <display-name>Web Services Adapter</display-name>
  <!-- List all of the servlet definitions here -->
  <servlet>
    <servlet-name>hr_servlet</servlet-name>
    <display-name>Web Services Adapter hr servlet</display-name>
    <servlet-class>com.progress.wsa.WsaServlet</servlet-class>
    <init-param>
      <param-name>InstallDir</param-name>
      <param-value>/progress/openedge</param-value>
    </init-param>
    <init-param>
      <param-name>instanceName</param-name>
      <param-value>hr</param-value>
    </init-param>
    <init-param>
      <param-name>propertyFileName</param-name>
      <param-value>/progress/openedge/properties/ubroker.properties</param-value>
    </init-param>
  </servlet>
  <servlet>
    <servlet-name>worktracking_servlet</servlet-name>
    <display-name>Web Services Adapter worktracking servlet</display-name>
    <servlet-class>com.progress.wsa.WsaServlet</servlet-class>
    <init-param>
      <param-name>InstallDir</param-name>
      <param-value>/progress/openedge</param-value>
    </init-param>
    <init-param>
      <param-name>instanceName</param-name>
      <param-value>worktracking</param-value>
    </init-param>
    <init-param>
      <param-name>propertyFileName</param-name>
      <param-value>/progress/openedge/properties/ubroker.properties</param-value>
    </init-param>
  </servlet>
  <servlet>
    <servlet-name>accntquery_servlet</servlet-name>
    <display-name>Web Services Adapter accounting-query servlet</display-name>
    <servlet-class>com.progress.wsa.WsaServlet</servlet-class>
    <init-param>
      <param-name>InstallDir</param-name>
      <param-value>/progress/openedge</param-value>
    </init-param>
    <init-param>
      <param-name>instanceName</param-name>
      <param-value>accntquery</param-value>
    </init-param>
    <init-param>
      <param-name>propertyFileName</param-name>
      <param-value>/progress/openedge/properties/ubroker.properties</param-value>
    </init-param>
  </servlet>
</web-app>
```
web.xml

```xml
<servlet>
  <servlet-name>accntg_servlet</servlet-name>
  <display-name>Web Services Adapter full accountingservlet</display-name>
  <servlet-class>com.progress.wsa.WsaServlet</servlet-class>
  <init-param>
    <param-name>InstallDir</param-name>
    <param-value>/progress/openedge</param-value>
  </init-param>
  <init-param>
    <param-name>instanceName</param-name>
    <param-value>accntg</param-value>
  </init-param>
  <init-param>
    <param-name>propertyFileName</param-name>
    <param-value>/progress/openedge/properties/ubroker.properties</param-value>
  </init-param>
</servlet>

<!-- List all of the servlet mappings here -->
<servlet-mapping>
  <servlet-name>hr_servlet</servlet-name>
  <url-pattern>/hr/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>worktracking_servlet</servlet-name>
  <url-pattern>/worktracking/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>accntgquery_servlet</servlet-name>
  <url-pattern>/accntgquery/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>accntg_servlet</servlet-name>
  <url-pattern>/accntg/*</url-pattern>
</servlet-mapping>

<!-- List all of the security contraints here -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>HR Web Services</web-resource-name>
    <url-pattern>/hr/*</url-pattern>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>Hrdept</role-name>
  </auth-constraint>
</security-constraint>
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Work-Tracking Web Services</web-resource-name>
    <url-pattern>/worktracking/*</url-pattern>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>Users</role-name>
  </auth-constraint>
</security-constraint>
```

<security-constraint>
  <web-resource-collection>
    <web-resource-name>Accounting-query Web Services</web-resource-name>
    <url-pattern>/accntgquery/</url-pattern>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>DeptMgr</role-name>
  </auth-constraint>
</security-constraint>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>Accounting Web Services</web-resource-name>
    <url-pattern>/accntg/</url-pattern>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>Acctdept</role-name>
  </auth-constraint>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Acme Company Web Services</realm-name>
</login-config>
</web-app>
Using the WSA Management Utility (WSAMAN)

This chapter provides an overview of the functions of the WSAMAN utility.
Overview of the WSAMAN utility

The WSAMAN utility is a command-line utility for administering Web Services Adapter (WSA) instances and Web services.

The functions provided by the WSAMAN utility duplicate their OpenEdge Explorer equivalents. For more information on OpenEdge Explorer equivalents, see the OpenEdge Explorer online help.

Syntax

Each function of the WSAMAN utility has syntax similar to the following:

Syntax

wsaman [ -option ] ... -function

[ -option ]...

One or more options, where each option consists of a name (for example, -appname) and a value (for example, wsainstance-name).

function

The name of the function (for example, getdefaults).

For more information on the functions of the WSAMAN utility, see Appendix B, “Command and Utility Reference.”

Functions for managing a WSA instance

Some functions of the WSAMAN utility allow you to manage a WSA instance. These are summarized in Table 8–1.

Table 8–1: WSA management functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSAMAN getdefaults</td>
<td>Displays the default Web service properties associated with a WSA instance</td>
</tr>
<tr>
<td>WSAMAN getprops (WSA)</td>
<td>Displays the current value of the properties that can be changed while the WSA is running</td>
</tr>
<tr>
<td>WSAMAN getstats (WSA)</td>
<td>Displays statistics for a WSA instance</td>
</tr>
<tr>
<td>WSAMAN list</td>
<td>Displays the list of Web service applications that have been deployed to the WSA instance</td>
</tr>
</tbody>
</table>
Overview of the WSAMAN utility

For more information on the functions of the WSAMAN utility, see Appendix B, “Command and Utility Reference.”

### Functions for managing a Web service

Other functions of the WSAMAN utility let you manage a Web service. These are summarized in Table 8–2.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSAMAN deploy</td>
<td>Deploys a Web service to a WSA instance.</td>
</tr>
<tr>
<td>WSAMAN disable</td>
<td>Makes a deployed Web service temporarily unavailable to incoming client requests.</td>
</tr>
<tr>
<td>WSAMAN enable</td>
<td>Makes a deployed Web service available to incoming client requests.</td>
</tr>
<tr>
<td>WSAMAN export</td>
<td>Creates a Web Service Definition (WSD) file on the local system from an existing Web service on an existing WSA instance.</td>
</tr>
</tbody>
</table>
Using the WSA Management Utility (WSAMAN)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSAMAN getprops (Service)</td>
<td>Displays the properties of a Web service. If the Web service is enabled, only the serviceFaultLevel and serviceLoggingLevel properties are displayed. If the Web service is disabled, all relevant properties of the Web service are displayed.</td>
</tr>
<tr>
<td>WSAMAN getstats (Service)</td>
<td>Displays the statistics for a Web service.</td>
</tr>
<tr>
<td>WSAMAN import</td>
<td>Deploys a previously exported Web service.</td>
</tr>
</tbody>
</table>
| WSAMAN query (Service)  | Displays the following information about a Web service:  
• Target NameSpace  
• Status  
• AppServer URL  
• Session model  
• WSDL Style/Use  
• Relevant Web service properties (regardless of whether the Web service is enabled or disabled) |
| WSAMAN resetprops      | Reinitializes a Web service’s friendlyName.props file to the current value of the WSA instance’s default.props file.                                                                     |
| WSAMAN resetstats (Service) | Resets the statistics of a Web service.                                                                                                                                                      |
| WSAMAN setprops (Service) | Sets one of the properties of a Web service.                                                                                                                                                 |
| WSAMAN undeploy        | Undeploys a Web service from a WSA instance.                                                                                                                                                  |
| WSAMAN update          | Lets you change a Web service’s deployment information (stored in the Web Service Mapping (WSM) file) without undeploying and deploying.                                                         |

**Note:** Progress Software Corporation recommends that update never be used on a production system.

For more information on the functions of the WSAMAN utility, see Appendix B, “Command and Utility Reference.”
Part IV

WebSpeed Administration

Chapter 9, Configuring WebSpeed in Windows
Chapter 10, Configuring WebSpeed on UNIX
Chapter 11, WebSpeed Dynamic Code-page Support
Chapter 12, Connecting WebSpeed to a Data Source
Chapter 13, WebSpeed Security
Chapter 14, Using Active Server Pages with WebSpeed
This chapter explains how to configure WebSpeed to run in Windows, as described in the following sections:

- WebSpeed configuration overview
- WebSpeed administration
- Setting up WebSpeed on the Web server machine
- Placing static files on the Web server
- Overview of the ubroker.properties file
- Starting the AdminService
- Setting up the WebSpeed environment
- Configuring WebSpeed components
- Starting the WebSpeed Transaction Server and NameServer
- Maintaining the WebSpeed Transaction Server and NameServer log files
- Configuring a WebSpeed Messenger-only installation
- Starting WebSpeed to test the configuration
- Testing your configuration
- Managing the WebSpeed Transaction Server
- Managing the WebSpeed Messenger
WebSpeed configuration overview

You must perform the following preliminary tasks before you can begin configuring the WebSpeed installation:

- 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 involved in the WebSpeed installation. This includes setting the appropriate environment variables and setting up your Web server.

For more information about these preliminary tasks, see *OpenEdge Getting Started: Installation and Configuration*.

Once you complete these preliminary tasks, you can begin to configure the WebSpeed components.

To configure the WebSpeed components:

1. Start the AdminService process on each machine. (See the “Starting the AdminService” section on page 9–20.)

2. Once the AdminService is running, you can use OpenEdge Explorer or Progress Explorer to create and modify NameServer and WebSpeed Transaction Server configurations. These configurations are in the properties file (`ubroker.properties`) on the machine where you installed the WebSpeed Transaction Server. (See the “OpenEdge Explorer or Progress Explorer” section on page 9–5.)

   You can also edit the `ubroker.properties` file manually by using a text editor. If you choose to use a text editor, you must have file system access to the file (see the “Overview of the `ubroker.properties` file” section on page 9–19).

   **Note:** You can use the `mergeprop` utility installed with OpenEdge to manually edit the `ubroker.properties` file. For information on using `mergeprop`, see *OpenEdge Getting Started: Installation and Configuration*.

   You can validate NameServer and WebSpeed configurations with the `NSCONFIG` and `WSCONFIG` validation utilities. These utilities must have access to the properties file. (See the “NSCONFIG” section on page B–15 and the “WSCONFIG” section on page B–45.)

3. Determine if you must set (or change any preset) WebSpeed environment variables. (See the “Setting up the WebSpeed environment” section on page 9–21.)

4. Use OpenEdge Explorer or Progress Explorer (or the `NSMAN` utility) to start up a NameServer instance that you configured to coordinate client access to one or more WebSpeed Transaction Server instances on your network. After the NameServer has started, use OpenEdge Explorer or Progress Explorer (or the `WTBMAN` utility) to start up each WebSpeed Transaction Server instance controlled by that NameServer. (See the “Starting the WebSpeed Transaction Server and NameServer” section on page 9–27.)

   After a WebSpeed Transaction Server starts up, it registers its location and the Application Services it supports with its controlling NameServer.
5. Set up the WebSpeed Messenger on your Web server. (See the “Setting up WebSpeed on the Web server machine” section on page 9–14.)

6. At any time after Step 5, you can verify that the AdminService, the NameServer, and the WebSpeed Transaction Server are running and test the configuration to confirm that you have set up WebSpeed correctly. You perform this verification with OpenEdge Explorer or Progress Explorer, the WebSpeed command-line utilities, or the WebSpeed Messenger Administration (WSMAdmin) page in a browser. (See the “Testing your configuration” section on page 9–32 for more information.)

7. You can shut down the AdminService process at any time on a WebSpeed or NameServer machine. If you shut down the AdminService while there are NameServer and WebSpeed Transaction Server instances still running, those instances are shut down as well.
WebSpeed administration

The WebSpeed administration framework consists of the following system administration components:

- **AdminService** — Provides access to OpenEdge servers to be administered on the local machine.
- **OpenEdge Explorer or Progress Explorer** — Allow local and remote administration and configuration of WebSpeed and other OpenEdge components.
- **Management utilities** — Allow administration from the command line of WebSpeed and other OpenEdge components.

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

The AdminService

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

The AdminService runs as a service in Windows and on UNIX. By default, it starts automatically.

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 choose 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.
OpenEdge Explorer or Progress Explorer

There are two graphical administration tools you can use to manage WebSpeed: OpenEdge Explorer and Progress Explorer. If both tools are available in your configuration, use OpenEdge Explorer. This tool provides access to the widest array of configuration and operational functions.

OpenEdge Explorer is a browser-based tool available for management of local and remote servers on any platform. Progress Explorer is a Windows-based tool available for local administration of Windows machines or remote administration of other platforms.

Both tools combine 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, BrokerConnect, Web Services, and databases. You can also use the tools to configure WebSpeed Messengers, start additional WebSpeed agents, or trim back running WebSpeed agents.

When you install WebSpeed, a sample WebSpeed Transaction Server (wsbroker1) and a sample NameServer (NS1) are installed automatically. To learn how to use your preferred tool, access the online help content of the tool.
Configuring WebSpeed with the OpenEdge Explorer

To configure WebSpeed in Windows, you can use OpenEdge Explorer.

To define and configure a WebSpeed instance:

1. Make sure the AdminServer is running on the host where you want to configure WebSpeed.
2. Open OpenEdge Explorer.
3. Connect to the AdminServer on your WebSpeed host.
4. To define a new WebSpeed instance or modify an existing WebSpeed configuration, do one of the following:
   - To define a new WebSpeed instance, select the WebSpeed folder in the tree view, and choose Add Resource, New, WebSpeed. Enter a unique name for the WebSpeed instance and click Save.
   - To modify an existing WebSpeed configuration, expand the WebSpeed folder in the tree view, select the WebSpeed instance you want to modify, and click Configuration.
5. Select a property category tab and set the properties as required. You can accept the default values, if they are appropriate for your application. You probably want to specify the properties under each category. See the online help for detailed information about each property.

The Broker category specifies properties of the WebSpeed broker. Expanding this category shows the following property subcategories:

- **General** — The Operating Mode is fixed stateless, and you probably want to specify a nondefault value for the TCP/IP port number where the WebSpeed broker listens for requests.
  
  If you want WebSpeed to start whenever you start the AdminServer, select the Auto start check box, and if you want to use a different working directory from the one specified during the WebSpeed product installation, you can also change it here.

- **Owner Information** — You can optionally provide Username and Password information for the user who owns the WebSpeed instance.

- **Controlling NameServer** — You must indicate whether you plan to use a NameServer to control WebSpeed access. If so, check the Register with NameServer box and select a controlling NameServer from the list of NameServer instances that you have already configured. Also specify how you want the NameServer to provide the host address of the WebSpeed broker for client connections.

- **AppService Name List** — You can either enter any names for the application services supported by WebSpeed or select the Supports default service check box if you want WebSpeed to support the default service for all client connections that do not specify an application service name. If you choose to use application service names, the default application service name is the name of the WebSpeed instance.
• **Logging Setting** — You can set the following WebSpeed broker logging options: specify a different pathname from the default for the broker log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous broker log file; specify a comma-separated list of logging entry types to be included in the broker log file, choosing from the valid values listed in the OpenEdge Explorer online help; set a file-size threshold that determines the point at which a new log file is created (0 = unlimited log file size); and specify the maximum number of broker log files to be kept (0 = unlimited number of log files retained). See *OpenEdge Development: Debugging and Troubleshooting* for detailed information on logging options.

• **Advanced Features** — You can specify the maximum number of client connections (*Maximum client instances*) that the WebSpeed broker can support at one time, the WebSpeed weight factor (*Priority weight*) for load balancing, the time between retries to register WebSpeed with the controlling NameServer, the time-out period for starting WebSpeed, the time-out period for WebSpeed to accept a client request, and the timeout period for the WebSpeed agent to trim its quota of WebSpeed agents between the maximum and minimum setting (see the *Agent* category). For more information on these options, see the OpenEdge Explorer online help.

The *Agent* category specifies properties of the WebSpeed agents. Expanding this category shows the following property subcategories:

• **General** — You can specify a pathname of the WebSpeed agent executable, startup parameters for the WebSpeed agent. For more information, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

For PROPATH, specify the semicolon-separated list of directories where WebSpeed can locate ABL procedures to execute. This setting overrides any PROPATH environment variable settings on the WebSpeed host when it starts up. Make sure that all of the ABL procedures (r-code or source) that you want WebSpeed to execute are located in one of these PROPATH directories. Otherwise, the procedure must be executed using its fully qualified pathname.

Specify the minimum and maximum TCP/IP port numbers that the WebSpeed agent can assign to WebSpeed agents that it starts up. (Check with your system administrator for appropriate ranges.)

Specify the mode for running applications during the current WebSpeed session using the *Server application mode*. The default mode is *Development*.

• **Logging Setting** — You can set the following WebSpeed agent logging options: specify a different pathname from the default for the server log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous server log file; specify a comma-separated list of logging entry types to be included in the server log file, choosing from the valid values listed in the OpenEdge Explorer online help; set a file-size threshold that determines the point at which a new log file is created (0 = unlimited log file size); and specify the maximum number of server log files to be kept (0 = unlimited number of log files retained). See *OpenEdge Development: Debugging and Troubleshooting* for detailed information on logging options.

• **Pool Range** — These settings determine the number of WebSpeed agents that the WebSpeed agent can start up and maintain for WebSpeed.
• **Advanced Features** — Specify the path a WebSpeed application uses relative to your Web server as the Application URL. You can specify settings for Default cookie path and Default cookie domain.

To specify the directory that WebSpeed agents use to upload text files, use **File upload directory**.

To allow the ABL debugger to run in the WebSpeed session, select the ABL debugger-enabled check box. Specify the names of any WebSpeed configuration procedures that you want WebSpeed to execute, and any parameters for the Startup procedure. For more information on debugging WebSpeed applications and on AppServer configuration procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

The options in the **SSL** category options define the security settings for an SSL-enabled WebSpeed instance. Note that a WebSpeed-enabled instance for SSL operation does not accept non-SSL client connections. For more information on SSL operations, see *OpenEdge Getting Started: Core Business Services*. Expanding this category shows the following property subcategories:

• **General** — If you check the Enable SSL client connections check box, select the alias for the private key/digital certificate entry (in the OpenEdge keystore) that you want to secure connections for this WebSpeed instance. Also enter and confirm the password for this private key and digital certificate. You need not enter a password if you choose to use the default_server certificate and its default password.

• **Advanced Features** — By default, caching is enabled for the SSL client session, and you can enter a time-out value that specifies the length of time (in seconds) that a disconnected session is held in the cache. During this specified interval, a connected client can resume its session. To disable session caching, check the box.

The **Messaging** category specifies properties for a ServerConnect process started by the application service running on this WebSpeed Transaction Server. It allows you to start a SonicMQ ServerConnect background process at startup. You can also set logging options for the SonicMQ ServerConnect process: specify a different pathname from the default for the broker log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous broker log file; specify a different pathname from the default for the server log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous server log file. See *OpenEdge Development: Debugging and Troubleshooting* for detailed information on logging options.

If you want to specify environment variables for WebSpeed execution, select the **Environment Variables** category. It allows you to enter name-value pairs for environment variable settings. Any values you set here override prior values set for the same environment variables in the operating system.

**Note:** Do not set the PROPATH variable in this category. Use the **Server General** category instead.

6. Choose the **OK** button to save the configuration.

For information about how to use OpenEdge Explorer to configure WebSpeed, see the OpenEdge Explorer online help.
Configuring WebSpeed with the Progress Explorer

To configure WebSpeed in Windows, you can use Progress Explorer.

To define and configure a WebSpeed instance:

1. Make sure the AdminServer is running on the host where you want to configure WebSpeed.

2. Open Progress Explorer.

3. Connect to the AdminServer on your WebSpeed host.

4. To define a new WebSpeed instance or modify an existing WebSpeed configuration, do one of the following:
   - To define a new WebSpeed instance, select the WebSpeed folder in Progress Explorer’s tree view, right-click, and choose New. Enter a unique name for the WebSpeed instance and click OK. Then open the property editor for the new instance by selecting the instance, right-clicking, and choosing Properties.
   - To modify an existing WebSpeed configuration, expand the WebSpeed folder in the tree view, select the WebSpeed instance you want to modify, and open the WebSpeed property editor.

The WebSpeed instance property editor shows a tree view of property categories on the left and the properties for the selected category on the right.

5. Select a property category and set the properties as required. You can accept the default values, if they are appropriate for your application. You probably want to specify the properties under each category. See the Progress Explorer online help for detailed information about each property.

The Broker category specifies properties of the WebSpeed broker. Expanding this category shows the following property subcategories:

- **General** — The Operating mode is fixed stateless, and you probably want to specify a nondefault value for the TCP/IP port number where the WebSpeed broker listens for requests.

  If you want WebSpeed to start whenever you start the AdminServer, select the Auto start check box, and if you want to use a different working directory from the one specified during the WebSpeed product installation, you can also change it here.

- **Owner Information** — You can optionally provide Username and Password information for the user who owns the WebSpeed instance.

- **Controlling NameServer** — You must indicate whether you plan to use a NameServer to control WebSpeed access. If so, check the Register With NameServer box and select a controlling NameServer from the list of NameServer instances that you have already configured. Also specify how you want the NameServer to provide the host address of the WebSpeed broker for client connections.
• **AppService Name List** — You can either enter any names for the application services supported by WebSpeed or select the **Supports default service** check box if you want WebSpeed to support the default service for all client connections that do not specify an application service name. If you choose to use application service names, the default application service name is the name of the WebSpeed instance.

• **Logging Setting** — You can set the following WebSpeed broker logging options: specify a different pathname from the default for the broker log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous broker log file; specify a comma-separated list of logging entry types to be included in the broker log file, choosing from the valid values listed in the Progress Explorer online help; set a file-size threshold that determines the point at which a new log file is created (0 = unlimited log file size); and specify the maximum number of broker log files to be kept (0 = unlimited number of log files retained). See *OpenEdge Development: Debugging and Troubleshooting* for detailed information on logging options.

• **Advanced Features** — You can specify the maximum number of client connections (**Maximum client instances**) that the WebSpeed broker can support at one time, the WebSpeed weight factor (**Priority weight**) for load balancing, the time between retries to register WebSpeed with the controlling NameServer, the time-out period for starting WebSpeed, the time-out period for WebSpeed to accept a client request, and the timeout period for the WebSpeed agent to trim its quota of WebSpeed agents between the maximum and minimum setting (see the **Agent** category). For more information on these options, see the Progress Explorer online help.

The **Agent** category specifies properties of the WebSpeed agents. Expanding this category shows the following property subcategories:

• **General** — You can specify a pathname of the WebSpeed agent executable, startup parameters for the WebSpeed agent. For more information, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

For **PROPATH**, specify the semicolon-separated list of directories where WebSpeed can locate ABL procedures to execute. This setting overrides any **PROPATH** environment variable settings on the WebSpeed host when it starts up. Make sure that all of the ABL procedures (r-code or source) that you want WebSpeed to execute are located in one of these **PROPATH** directories. Otherwise, the procedure must be executed using its fully qualified pathname.

Specify the minimum and maximum TCP/IP port numbers that the WebSpeed agent can assign to WebSpeed agents that it starts up. (Check with your system administrator for appropriate ranges.)

Specify the mode for running applications during the current WebSpeed session using the **Agent Application Mode**. The default mode is **Development**.
• **Logging Setting** — You can set the following WebSpeed agent logging options:
  specify a different pathname from the default for the server log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous server log file; specify a comma-separated list of logging entry types to be included in the server log file, choosing from the valid values listed in the Progress Explorer online help; set a file-size threshold that determines the point at which a new log file is created (0 = unlimited log file size); and specify the maximum number of server log files to be kept (0 = unlimited number of log files retained). See *OpenEdge Development: Debugging and Troubleshooting* for detailed information on logging options.

• **Pool Range** — These settings determine the number of WebSpeed agents that the WebSpeed agent can start up and maintain for WebSpeed.

• **Advanced Features** — Specify the path a WebSpeed application uses relative to your Web server as the **Application URL**. You can specify settings for **Default Cookie Path** and **Default Cookie Domain**.

To specify the directory that WebSpeed agents use to upload text files, use **File Upload Directory**.

To allow the ABL debugger to run in the WebSpeed session, select the ABL debugger-enabled check box. Specify the names of any WebSpeed configuration procedures that you want WebSpeed to execute, and any parameters for the Startup procedure. For more information on debugging WebSpeed applications and on AppServer configuration procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

The options in the **SSL** category options define the security settings for an SSL-enabled WebSpeed instance. Note that a WebSpeed-enabled instance for SSL operation does not accept non-SSL client connections. For more information on SSL operations, see *OpenEdge Getting Started: Core Business Services*. Expanding this category shows the following property subcategories:

• **General** — If you check the **Enable SSL Client Connections** check box, select the alias for the private key/digital certificate entry (in the OpenEdge keystore) that you want to secure connections for this WebSpeed instance. Also enter and confirm the password for this private key and digital certificate. You need not enter a password if you choose to use the *default_server* certificate and its default password.

• **Advanced Features** — By default, caching is enabled for the SSL client session, and you can enter a time-out value that specifies the length of time (in seconds) that a disconnected session is held in the cache. During this specified interval, a connected client can resume its session. To disable session caching, check the box.
The **Messaging** category specifies properties for a ServerConnect process started by the application service running on this WebSpeed Transaction Server. It allows you to start a SonicMQ ServerConnect background process at startup. You can also set logging options for the SonicMQ ServerConnect process: specify a different pathname from the default for the broker log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous broker log file; specify a different pathname from the default for the server log file; specify the level of logging detail; control whether the logging for a session appends to or overwrites the previous server log file. See *OpenEdge Development: Debugging and Troubleshooting* for detailed information on logging options.

If you want to specify environment variables for WebSpeed execution, select the **Environment Variables** category. It allows you to enter name-value pairs for environment variable settings. Any values you set here override prior values set for the same environment variables in the operating system.

**Note:** Do not set the **PROPATH** variable in this category. Use the **Server General** category instead.

6. Choose the **OK** button to save the configuration.

For information about how to use Progress Explorer to configure WebSpeed, see the Progress Explorer online help.

**WebSpeed command-line utilities**

The **WTBMAN** and **WSCONFIG** command-line utilities provide the ability to manage and query the status of running WebSpeed Transaction Servers and serve as diagnostic tools to display a broker’s configuration.

**WTBMAN utility**

You can 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.

For more information about the **WTBMAN** utility, see the “**WTBMAN**” section on page B–47.
**WSCONFIG utility**

You can use the `WSCONFIG` utility to validate existing WebSpeed Transaction Server or WebSpeed Messenger configurations. The `WSCONFIG` utility reads the `ubroker.properties` file for validation.

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

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

For more information about the `WSCONFIG` utility, see the “**WSCONFIG**” section on page B–45.

**NameServer command-line utilities**

Running as an independent process within the WebSpeed architecture, the NameServer supports load-balancing functionality (when the OpenEdge Server Enterprise is installed) and location transparency. The `NSMAN` and `NSCONFIG` command-line utilities manage, query the running status of, and display the NameServer configuration.

**Note:** The NameServer can simultaneously support the WebSpeed Transaction Server, AppServers, and DataServers.

**NSMAN utility**

You can use the `NSMAN` utility to control the operation of a configured NameServer. The utility provides the ability to start a NameServer, query a NameServer status, and shut down a NameServer. For more information about the `NSMAN` utility, see the “**NSMAN**” section on page B–17.

**NSCONFIG utility**

You can use the `NSCONFIG` utility to query the current configuration of an existing NameServer and to view all the option values for a specific NameServer. This utility is a diagnostic tool and can be helpful when you are attempting to validate and resolve configuration settings.

The `NSCONFIG` configuration command runs locally only, on the machine where the NameServer is installed. The utility does not run across the network.

For more information about the `NSCONFIG` utility, see the “**NSCONFIG**” section on page B–15.


**Setting up WebSpeed on the Web server machine**

This section describes how to set up WebSpeed on your Web server machine. Keep in mind that the machine running your Web server can be the same machine running the Transaction Server. However, you might want to use more than one machine, depending on the size of your WebSpeed application, the computing power of your Web server machine, and security considerations.

---

**Note:** Since WebSpeed can run on a wide range of Web servers, it is not possible to provide specific instructions here for configuring your Web server. For specific information about your Web server, see your Web server documentation.

---

**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 9–1.

### Table 9–1: 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>WSASP¹</td>
<td>wsasp.dll</td>
</tr>
<tr>
<td>Netscape</td>
<td>NSAPI</td>
<td>wsnsa.dll</td>
</tr>
<tr>
<td>CGI-compatible</td>
<td>CGI</td>
<td>cgiip.exe</td>
</tr>
</tbody>
</table>

¹ The WSASP Messenger calls WebSpeed applications from an Active Server Page. It cannot coexist with the ISAPI Messenger.

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.

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.

If you install the WebSpeed Transaction Server on an operating system that is different from the operating system that runs your Web server, you need to go to download a compatible Messenger. Access the Progress Software Download Center and log in to your account at http://www.progress.com/esd. Navigate to the WebSpeed Messenger download page by selecting Download Software and then Progress® Deployment Components. 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.

Configuring Netscape Web servers for NSAPI Messengers

To configure a Netscape Web server to work with the WebSpeed NSAPI Messenger, you must edit the Netscape Web server configuration file (obj.conf).

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. Table 9–2 describes the changes you must make to obj.conf.

<table>
<thead>
<tr>
<th>New Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init fn=load-modules shlib=&quot;pathname&quot;</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>funcs=WSNSAinit,WSNSAdefault,WSNSAshutdown,WSNSAWebSpeedCheck</td>
<td></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>
</tbody>
</table>
Configuring WebSpeed in Windows

Table 9–2: Netscape Web server configuration (2 of 2)

<table>
<thead>
<tr>
<th>New Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NameTrans fn=WSNSAWebSpeedCheck</td>
<td>This NameTrans line must appear <strong>before</strong> any existing NameTrans commands. You must add it within the block of commands delimited by <code>&lt;Object name=default&gt;</code> and <code>&lt;/Object&gt;</code>. 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):

```plaintext
Init ...
Init ...
# The following directive is a single line; it contains no line breaks
Init fn=load-modules shlib="c:\Program Files\OpenEdge\bin\wsnsa.dll"
  func=WSNSAinit,WSNSAdefault,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=WSNSAdefault
  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.
Placing static files on the Web server

Some of the HTML pages in your Web application might never change. These types of pages are called static HTML pages; they are never processed by WebSpeed. Java class files that your Web application uses are also considered static files. WebSpeed Workshop and WebSpeed applications rely on the static HTML and Java class files in \install-path\WebSpeed. Static Web files must be located under your Web server’s document root directory. Alternately, you can configure your Web server to look for static files in the directory of your choice.

If your WebSpeed application uses static HTML files, you should place them on the Web server machine in a subdirectory of the Web server’s document root directory (or consult your Web server documentation for information about other options, such as creating an additional document root directory). You must place image files (.gif or .jpeg), audio files (.au), and video files (.mpeg, .mpg) on the Web server machine. These files cannot be served directly by WebSpeed. You must place these files on the Web server machine.

WebSpeed objects or compiled code must be accessible and visible to the WebSpeed agents. Make the files accessible by placing them on the machine where your Transaction Server runs. You can make a file visible to the WebSpeed agents by adding its directory name to the Transaction Server’s PROPATH.
Overview of the ubroker.properties file

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

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

You use OpenEdge Explorer to create and configure instances of the WebSpeed Transaction Server or the NameServer on Windows or remote UNIX platforms. It is possible to do this by editing the ubroker.properties file manually. See the “Overview of the ubroker.properties file” section on page 10–13 for more information on the ubroker.properties file.

Note: You can use the mergeprop utility installed with OpenEdge to manually edit the ubroker.properties file. For information on using mergeprop, see OpenEdge Getting Started: Installation and Configuration.
Starting the AdminService

The AdminService provides a command and control interface for the WebSpeed Transaction Server, NameServer, and other components. The AdminService must start before you can start WebSpeed.

The AdminService runs as a service in Windows and starts automatically by default.

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 choose 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.
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 in Windows by using OpenEdge Explorer or the WebSpeed command-line utilities.

Setting environment variables

Environment variables required by WebSpeed are set during installation in the unbroker.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 Explorer or by editing the WebSpeed property file, unbroker.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 unbroker.properties file. 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 (unbroker.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 nondefault 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 log files to reside, you can specify the location for each in OpenEdge Explorer or by directly editing the ubroker.properties file. For more information, see the “Configuring WebSpeed components” section on page 9–23.

Because log files receive the WebSpeed and NameServer startup and shutdown messages, OpenEdge system messages, and trace messages, the file 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 file with a text editor. You might want to archive the file contents as you do it. For more information on maintaining log files, see the “Maintaining the WebSpeed Transaction Server and NameServer log files” section on page 9–29.
Configuring WebSpeed components

You can use OpenEdge Explorer to create and configure the NameServer and the WebSpeed broker. You can use OpenEdge Explorer or the command-line utilities to manage the configuration.

For more information about creating, configuring, and managing with OpenEdge Explorer, see the online help.

For information about managing with the command-line utilities, see the “WebSpeed command-line utilities” section on page 9–12.

Eliminating the NameServer

Use of the NameServer is optional. The following are reasons why you might want to eliminate the NameServer from your configuration:

- The NameServer allows for location transparency and load balancing. Simple or static configurations might not require those features.
- The communications protocol employed by the NameServer is 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.

To eliminate the NameServer using OpenEdge Explorer:

1. Select your broker and click Configuration.
2. Select Controlling NameServer under the Broker tab and click Edit.
3. Deselect the Register With NameServer check box.
4. Choose the Save button.
5. Select your Messenger and click Configuration.
6. Click the Controlling NameServer tab and click Edit.
7. Deselect the Register with NameServer check box.
8. Type the hostname and port number of the broker.
9. Choose the Save button.
You can also eliminate the NameServer by directly editing the ubroker.properties file, although using OpenEdge Explorer is less error prone.

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

1. Open `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 only access one WebSpeed Transaction Server (broker). One of the advantages of using the NameServer is that you can run multiple brokers.
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 your product includes load balancing (for example, the WebSpeed Enterprise Transaction Server) or you have installed the load-balancing option for a product that requires it (for example, the AppServer), 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 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, Table 9–3 shows the effect of weight factors specified for three WebSpeed Transaction Server instances registered for the same application service.

<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 Table 9–4.

<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</td>
</tr>
<tr>
<td>WS2</td>
<td>2</td>
<td>2/7</td>
</tr>
<tr>
<td>WS3</td>
<td>3</td>
<td>3/7</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*. 
Starting the WebSpeed Transaction Server and NameServer

This section outlines the steps to begin the WebSpeed Transaction Server and NameServer for developing and deploying WebSpeed applications in Windows.

To begin to develop or deploy applications in Windows:

1. The AdminService must be running. If the AdminService is not running, you must start it (see the “Starting the AdminService” section on page 9–20).

2. Start an existing NameServer or create a new NameServer instance. You can create and start a NameServer by using 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 on UNIX.

   If you are using OpenEdge Explorer, see the online help for information about creating and starting an instance. If you are editing the ubroker.properties file, see the “Editing the ubroker.properties file” section on page 10–13.

   To start a local instance of the NameServer (NS1) from the command line, use the following command:

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

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

   ```
   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 under which the NameServer will run.

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 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.

   See the OpenEdge Explorer online help for information about creating and starting an instance.
To start a local instance of the WebSpeed Transaction Server from the command line, use the following command:

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

Where `wsbroker1` is the name of the default WebSpeed broker.

**Note:** The WebSpeed Transaction Server consists of a broker and agents. 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:

```
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, OpenEdge Explorer prompts you for a user name (if you do not supply it) and password.

By using either OpenEdge Explorer 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 Explorer online help, the “Using OpenEdge Explorer to check status” section on page 9–32, and the “Managing the WebSpeed Transaction Server” section on page 9–34.
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. Thus, the information in these log files can be useful when you perform routine maintenance or troubleshooting.

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 the “Configuring WebSpeed and NameServer log files” section on page 9–21.
Configuring a WebSpeed 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. Instead, you can install the Web server and the WebSpeed Messengers together on a different machine if you want.

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.
Starting WebSpeed to test the configuration

After you set up the WebSpeed Transaction Server machine and the Web server machine, you are ready to run WebSpeed and test your configuration.

To run WebSpeed and test your configuration:

1. Make sure the AdminService is running.
2. Make sure the NameServer is running.

For more information, see the “Starting the WebSpeed Transaction Server and NameServer” section on page 9–27.
Testing your configuration

To test your configuration:

- Use OpenEdge Explorer to view a detailed status
- Use the -query option of the NSMAN and WTBMAN utilities to check the status of components
- View the Messenger Administration page (WSMAdmin)
- Run the status.p procedure from a Web browser

Using OpenEdge Explorer to check status

You can check the WebSpeed configuration status from OpenEdge Explorer. Select the WebSpeed broker and click Status.

Using the query option to check status

You can test your configuration and check its status by running the -query option on the NameServer and the WebSpeed broker.

To query a local NameServer, use the NSMAN utility, which is installed in install-path\bin, as follows:

```
nsman -name NameServer -query
```

Where NameServer is the name of the NameServer that you want to query.

To query a remote NameServer, use the following command:

```
nsman -name NameServer -host host -port port -user user -query
```

To query a local WebSpeed broker, use the WTBMAN utility shown, which is installed in install-path\bin:

```
wtbman -name broker -query
```

Where broker is the name of the WebSpeed broker that you want to query. You can have more than one broker running on a single machine.

To query a remote WebSpeed broker, use the following command:

```
wtbman -name broker -host host-name -port port -user user-name -query
```
The query reports on the broker’s pool of WebSpeed agents. For each agent, it lists a process ID, its port number, its status, how many requests it has serviced, when it started, and when its status changed.

The following are the agent status types:

- **AVAILABLE** — The agent is available to execute application logic.
- **BUSY** — The agent is actively executing application logic (Web object) for a Web user.
- **LIMBO** — The agent is in a transitional state. If the status persists, it indicates an error condition.
- **LOCKED** — The agent is dedicated to a particular Web browser and is only available to the browser whose application locks it.
- **STARTING** — The broker has launched the agent, but the agent has not yet initialized.

**Viewing the Messenger Administration page**

You can test to see if the Web server is fully operational by pointing your Web browser to the WSMAdmin page and following the directions. The WSMAdmin page allows you to perform a round-trip test of your entire WebSpeed configuration or to test each component or leg separately.

See the “Managing the WebSpeed Messenger” section on page 9–36 for the URL for the WSMAdmin page.

**Running the status.p procedure**

You can run the status.p procedure from a Web browser. The status.p procedure is installed into the `install-path\src\web\examples` directory. Running this procedure lets you check that WebSpeed and your Web server are fully operational. If you successfully run this procedure, it means that you configured the WebSpeed run-time environment correctly. If you then have errors running your own application, you can be confident that the problem is specific to your application files.

To run the status.p procedure from a Web browser, enter a URL using the following format:

```
http://hostname/script-dir/Messenger/WService=broker/src/web/examples/status.p
```
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 enables you to 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:

```bash
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:

```bash
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
```
Managing the WebSpeed Messenger

Sometimes, 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
```

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.
This chapter explains how to configure WebSpeed to run on UNIX, as described in the following sections:

- WebSpeed configuration overview
- WebSpeed administration
- Setting up WebSpeed on the Web server machine
- Placing static files on the Web server
- Overview of the ubroker.properties file
- Starting the AdminServer
- Setting up the WebSpeed environment
- Configuring WebSpeed components
- Starting the WebSpeed Transaction Server and NameServer
- Maintaining the WebSpeed Transaction Server and NameServer log files
- Configuring a WebSpeed Messenger-only installation
- Starting WebSpeed to test the configuration
- Testing your configuration
- Managing the WebSpeed Transaction Server
- Managing the WebSpeed Messenger
WebSpeed configuration overview

You must perform the following preliminary tasks before you can begin configuring the WebSpeed installation:

- 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 involved in the WebSpeed installation. This includes setting the appropriate environment variables and setting up your Web server.

For more information about these preliminary tasks, see OpenEdge Getting Started: Installation and Configuration.

Once you complete these preliminary tasks, you can begin to configure the WebSpeed components.

To configure the WebSpeed components:

1. Start the AdminServer process on each machine by using the PROADSV utility. (See the “Starting the AdminServer” section on page 10–16.)

2. Once the AdminServer is running, you can then use OpenEdge Explorer or a text editor to create and modify NameServer and WebSpeed Transaction Server configurations. These configurations are in the properties file (ubroker.properties) on the machine where you installed the WebSpeed Transaction Server. You can use OpenEdge Explorer remotely from Windows to access configurations installed on UNIX.

   You can also edit the ubroker.properties file manually using a text editor. If you choose to use a text editor, you must have file system access to the file (see the “Overview of the ubroker.properties file” section on page 10–13).

   **Note:** You can use the mergeprop utility installed with OpenEdge to manually edit the ubroker.properties file. For information on using mergeprop, see OpenEdge Getting Started: Installation and Configuration.

   You can validate NameServer and WebSpeed configurations using the validation utilities, NSCONFIG and WSCONFIG. These utilities must have access to the properties file (see the “NSCONFIG” section on page B–15 and the “WSCONFIG” section on page B–45).

3. Determine if you need to set (or change any preset) WebSpeed environment variables (see the “Setting up the WebSpeed environment” section on page 10–17).

4. Use the NSMAN utility (or OpenEdge Explorer remotely) to start up a NameServer instance that you configured to coordinate client access to one or more WebSpeed Transaction Server instances on your network. After the NameServer has started, use the WTBMAN utility (or OpenEdge Explorer remotely) to start up each WebSpeed Transaction Server instance controlled by that NameServer (see the “Starting the WebSpeed Transaction Server and NameServer” section on page 10–20).

   After a WebSpeed Transaction Server starts up, it registers its location and the Application Services it supports with its controlling NameServer.
5. Set up the WebSpeed Messenger on your Web server (see the “Setting up WebSpeed on the Web server machine” section on page 10–6).

6. At any time after Step 5, you can validate that the AdminServer, the NameServer, and the WebSpeed Transaction Server are running and test the configuration to confirm that you have set up WebSpeed correctly. You perform this validation by using the WebSpeed command-line utilities, OpenEdge Explorer, or the WebSpeed Messenger Administration (WSMAdmin) page in a browser (see the “Testing your configuration” section on page 10–25 for more information).

7. You can shut down the AdminServer process at any time on the WebSpeed machine. If you shut down the AdminServer while there are NameServer and WebSpeed Transaction Server instances still running, those instances are shut down as well.
WebSpeed administration

The WebSpeed administration framework consists of the following system administration components:

- The AdminServer, which provides remote administrative access
- OpenEdge Explorer, which allows 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

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

The AdminServer

The AdminServer, the core of the common administration framework, supports the managing of WebSpeed and other OpenEdge products (for example, NameServer, database, DataServer) and is also used by other OpenEdge processes.

By using the PROADSV utility, you can start up and shut down the AdminServer. Remember that the AdminServer serves as a connection point for both local and remote services for configuration and administration.

OpenEdge Explorer or Progress Explorer

There are two graphical administration tools you can use to manage WebSpeed: OpenEdge Explorer and Progress Explorer. If both tools are available in your configuration, use OpenEdge Explorer. This tools provides access to the widest array of configuration and operational functions.

OpenEdge Explorer is browser-based tool available for management of local and remote servers on any platform. Progress Explorer is a Windows-based tool available for local administration of Windows machines or remote administration of other platforms.

Both tools combine 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, BrokerConnect, Web Services, and databases. You can also use the tools to configure WebSpeed Messengers, start additional WebSpeed agents, or trim back running WebSpeed agents.

When you install WebSpeed, a sample WebSpeed Transaction Server (wsbroker1) and a sample NameServer (NS1) are installed automatically. To learn how to use your preferred tool, access the online help content of the tool.

WebSpeed command-line utilities

The WTBMAN and WSCONFIG command-line utilities provide the ability to manage and query the status of running WebSpeed Transaction Servers and serve as diagnostic tools to display a Transaction Server’s configuration.
WTBMAN utility
You can 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.

For more information about the WTBMAN utility, see the “WTBMAN” section on page B–47.

WSCONFIG utility
You can use the WSCONFIG utility to validate existing WebSpeed Transaction Server or WebSpeed Messenger configurations. The WSCONFIG utility reads the ubroker.properties file for validation.

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

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

For more information about the WSCONFIG utility, see the “WSCONFIG” section on page B–45.

NameServer command-line utilities
Running as an independent process within the WebSpeed architecture, the NameServer supports load-balancing functionality (when the WebSpeed Enterprise Transaction Server is installed) and location transparency. The NSMAN and NSCONFIG command-line utilities manage, query the running status of, and display the NameServer configuration.

**Note:** The NameServer can simultaneously support the WebSpeed Transaction Server, AppServers, and DataServers.

NSMAN utility
You can use the NSMAN utility to control the operation of a configured NameServer. The utility provides the ability to start a NameServer, query a NameServer status, and shut down a NameServer.

For more information about the NSMAN utility, see the “NSMAN” section on page B–17.

NSCONFIG utility
You can use the NSCONFIG utility to query the current configuration of an existing NameServer and can be used to view all the option values for a specific NameServer. This utility is a diagnostic tool and can be helpful when you are attempting to validate and resolve configuration settings.

The NSCONFIG configuration command only runs locally, on the machine where the NameServer is installed. The utility does not run across the network.

For more information about the NSCONFIG utility, see the “NSCONFIG” section on page B–15.
Configuring WebSpeed on UNIX

Setting up WebSpeed on the Web server machine

This section describes how to set up WebSpeed on your Web server machine. Keep in mind that the machine running your Web server can be the same machine running the Transaction Server. However, you might want to use more than one machine, depending on the size of your WebSpeed application, the computing power of your Web server machine, and security considerations.

Note: Since WebSpeed can run on a wide range of Web servers, it is not possible to provide specific instructions here for configuring your Web server. For specific information about your Web server, see your Web server documentation.

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 10–1.

Table 10–1: UNIX WebSpeed Messengers

<table>
<thead>
<tr>
<th>Web server type</th>
<th>Messenger executable</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSAPI (Netscape)</td>
<td>wsnsa.dll</td>
</tr>
<tr>
<td>CGI-compatible</td>
<td>cgiip</td>
</tr>
</tbody>
</table>

When you installed WebSpeed, you provided information about your Web server. If you selected CGI, the installation utility installed the CGI Messenger script, wspd.cgi.sh, into the directory you specified as your Web server scripts directory. The NSAPI executable resides and runs from the OpenEdge install-path/bin directory.

Note: You must restart the 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.

If you install the WebSpeed Transaction Server on an operating system that is different from the operating system that runs your Web server, you need to go to download a compatible Messenger. Access the Progress Software Download Center and log in to your account at http://www.progress.com/esd. Navigate to the WebSpeed Messenger download page by selecting Download Software and then Progress® Deployment Components. This is necessary because you must install a Messenger that is compatible with the Web server’s operating system. For example, if your Web server is CGI-compatible and runs in Windows, you must install a CGI Messenger that can run in Windows.
Configuring the Netscape Web server for the NSAPI Messenger

You must perform the following tasks to configure a Web server to work with the WebSpeed NSAPI Messenger:

- Edit the Netscape Web server configuration file (obj.conf)
- Edit the Netscape Web server start file

Editing the Netscape Web server configuration file

A 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, refer to 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. Table 10–2 describes the changes you must make to `obj.conf`.

Table 10–2: Netscape Web server configuration

<table>
<thead>
<tr>
<th>New line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Init fn=load-modules shlib=&quot;pathname&quot;</code> <code>funcs=WSNSAinit,WSNSAdefault, WSNSAshutdown,WSNSAwebspeedCheck</code></td>
<td>This pathname is the absolute path to the Messenger shared object, <code>wsnsa.dll</code>. The Init line must appear after any existing Init commands. This line informs the Web server that it must load the Messenger shared object at the end of the initialization process, and that the named functions are external entry points within the shared object.</td>
</tr>
<tr>
<td><code>Init fn=WSNSAinit</code></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 shared object.</td>
</tr>
<tr>
<td><code>NameTrans fn=WSNSAwebspeedCheck</code></td>
<td>This NameTrans line must appear before any existing NameTrans commands. You must add it within the block of commands delimited by <code>&lt;Object name=default&gt;</code> and <code>&lt;/Object&gt;</code>. 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):

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLC</td>
<td>The WebSpeed Messenger installation path</td>
</tr>
<tr>
<td>PROMSGS</td>
<td>The pathname of the PROMSGS file: \install-path/promsgs</td>
</tr>
<tr>
<td>PROCFG</td>
<td>The pathname of the configuration file: \install-path/progress.cfg</td>
</tr>
<tr>
<td>WRKDIR</td>
<td>The pathname of your working directory</td>
</tr>
<tr>
<td>LD_LIBRARY_PATH</td>
<td>Shared library search path (on Solaris, Compaq Tru64 UNIX and Linux only)</td>
</tr>
<tr>
<td>SHLIB_PATH</td>
<td>Shared library search path (on HP-UX only)</td>
</tr>
<tr>
<td>LIBPATH1</td>
<td>Shared library search path (on AIX only)</td>
</tr>
</tbody>
</table>

1. On AIX only, you need to add /usr/lib to the shared library search path environment variable.
Setting up WebSpeed on the Web server machine

The following is a sample start file for AIX that includes the necessary WebSpeed information:

```
#!/bin/sh

LIBPATH=/usr/lib:$LIBPATH export LIBPATH
...
DLC=/usr/dlc; export DLC
PROMSGS=/$DLC/promsgs; export PROMSGS
PROCFG=$DLC/progress.cfg; export PROCFG
WRKDIR=/usr/workdir export WRKDIR
```

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 running. Consult your Web server documentation for more information about running the Web server.

Configuring a CGI Messenger

Configuring a Messenger to access a CGI-compatible Web server requires only that you create a script file that the Web server uses to invoke the Messenger executable, cgiip. You do not have to change the configuration of your Web server. The script file sets WebSpeed environment variables and invokes the Messenger executable, cgiip.

You can find a sample file that you can use as the basis for a shell script to configure the script file at install-path/bin/wspd_cgi.sh.
The following is an edited version of \textit{install-path/bin/wspd_cgi.sh}:

```bash
# Determine the correct directory where the Messenger
# is installed from either the tailored name or existing value of $DLC.
for what_dlc in "/usr1/pmccorma/wsrt" "$DLC"
do
    [ ! -f "${what_dlc}/bin/cgiip" ] && continue
    DLC=${what_dlc}
    export DLC
    break
done

# Set PROMSGS (if not set)
PROMSGS=${PROMSGS-$DLC/promsgs}; export PROMSGS

# Set the user working directory - this is a tailored value
WRKDIR=${WRKDIR-"/usr1/pmccorma/wrk"}; export WRKDIR

# option 1 using host_name and port_num
# $DLC/bin/cgiip pegasus 5001
# option 2 using a service name defined in $DLC/ubroker.properties
# $DLC/bin/cgiip -i wsbroker1
# option 3 the "defaultService" defined in $DLC/ubroker.properties
# $DLC/bin/cgiip
# option 4 using a specific properties file name
# $DLC/bin/cgiip -i wsbroker1 -f ./mybroker.properties
# option 5 using a specific properties file name with the "defaultService"
# $DLC/bin/cgiip -f ./mybroker.properties
```

The \texttt{wspd_cgi.sh} script provides and documents additional options for invoking the Messenger executable. You can, for example, specify the default WebSpeed service or use the \texttt{-f} option to specify a named configuration file. Table 10–4 describes the various options.

**Table 10–4: Options for invoking the CGI Messenger**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{cgiip}</td>
<td>The Messenger uses the Transaction Server that you defined as the Default Service in the \texttt{ubroker.properties} file.</td>
</tr>
<tr>
<td>\texttt{cgiip host-name port-num}</td>
<td>The Messenger uses the Transaction Server whose host name and port number you specify.</td>
</tr>
<tr>
<td>\texttt{cgiip -i WS-service}</td>
<td>The Messenger uses the Transaction Server service whose name you specify.</td>
</tr>
<tr>
<td>\texttt{cgiip -f cnf-file}</td>
<td>The Messenger uses the configuration file that you specify. This option overrides the \texttt{ubroker.properties} setting.</td>
</tr>
</tbody>
</table>

**Note:** Specifying a WebSpeed service name in a URL overrides any settings in your Messenger script file.
You can edit the script if you would like to see what information `cgiip` is sending to the browser for debugging purposes (including HTTP headers). The following is a sample script that shows how you might do this:

```
# save CGI environment
env >/tmp/webapp.env
# save application output
outfile=/tmp/webapp.out
/usr/dlc/bin/cgiip -i WSbroker 2>&1 | tee $outfile
```

After a request from a Web browser initiates this script, the script saves all of the CGI environment information in a file named `webapp.env`. The script sends its output not only to the browser but also to a file named `/tmp/webapp.out`. You can look at `/tmp/webapp.out` to see what was sent to the browser. The technique is useful for pinpointing problems that might be related more to the server than to your application. A clue that this might be the case is if your browser receives no output from your application and displays server errors.

Remember that this script executes every time a browser makes a request. Thus, if multiple browsers make multiple simultaneous requests, the agents servicing the requests can overwrite each other’s output to the file. For this reason, you should use this technique only when you are debugging your application (typically with a controlled number of agents and browsers).

**Where to place the Messenger script file**

Typically, a Messenger script file, such as `wspd_cgi.sh`, resides in the `cgi-bin` 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. You can also designate whether the executable files require a specific extension (such as `.cgi`) or can use any extension. See your Web server documentation for details on how to configure a scripts directory and CGI programs.

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

Many UNIX Web servers allow CGI programs to reside in directories other than `cgi-bin`. However, some Web servers might require a specific extension. See your Web server documentation for more information.
Placing static files on the Web server

Some of the HTML pages in your Web application may never change. These types of pages are called static HTML pages; they are never processed by WebSpeed. Java class files that your Web application uses are also considered static files. WebSpeed Workshop and WebSpeed applications rely on the static HTML and Java class files in `install-path/webspeed`. Static web files must be located under your Web server’s document root directory. Alternately, you can configure your Web server to look for static files in the directory of your choice.

If your WebSpeed application uses static HTML files, you should place them on the Web server machine in a subdirectory of the Web server’s document root directory (or consult your Web server documentation for information about other options, such as creating an additional document root directory). You must place image files (`.gif`), audio files (`.au`), and video files (`.mpeg`, `.mpg`) on the Web server machine. These files cannot be served directly by WebSpeed. You must place these files on the Web server machine.

WebSpeed objects or compiled code must be accessible and visible to the WebSpeed agents. Make the files accessible by placing them on the machine where your Transaction Server runs. You can make a file visible to the WebSpeed agents by adding its directory name to the Transaction Server’s PROPATH.
Overview of the 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 Explorer use this information when working with instances of all processes.

The ubroker.properties file resides in the install-path/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.

The ubroker.properties file structure

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.

For more information on the structure of the ubroker.properties file, see OpenEdge Getting Started: Installation and Configuration.

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

If you edit the configuration by hand, without OpenEdge Explorer, note the following:

- 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 Explorer available, use it to make all changes to this file on UNIX.
- For complete definitions of all the properties and detailed information on how to set them, see the comments included in the installed 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** — 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. Set only one of the Transaction Servers as the default.
- **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 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 cut and paste the value after “uuid=”.

- **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.

**Note:** The srvrStartupParam property is a dynamic property. Any changes to this property only affects new agents started after the value changes because the change affects how the agent is started. Existing (running) brokers and agents are not updated with the new value.
– **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 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 a complete overview of all the sections in the ubroker.properties file and more information on how to edit the file, see the comments in the file, or see *OpenEdge Getting Started: Installation and Configuration*. You can use the mergeprop utility installed with OpenEdge to manually edit the ubroker.properties file. For information on using mergeprop, see *OpenEdge Getting Started: Installation and Configuration*. For more information on the configuration utilities, see Appendix B, “Command and Utility Reference.”
Starting the AdminServer

The AdminServer provides a command and control interface for the WebSpeed Transaction Server, NameServer, and other components. You must start the AdminServer before you can start WebSpeed.

The PROADSV utility provides you with the ability to start up and shut down the AdminServer on UNIX. This command runs locally on the AdminServer machine.

Start the AdminServer by running the following command:

```
proadsv -start
```

For more information on using the PROADSV utility, see the “PROADSV” section on page B–19.
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 UNIX by using the WebSpeed command-line utilities.

Alternately, you can use OpenEdge Explorer.

Setting environment variables

Environment variables required by WebSpeed are set in the ubroker.properties file during installation. You might also 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 Explorer or by editing the WebSpeed property file, ubroker.properties.

PROPATH and other standard OpenEdge environment variables

When you install the WebSpeed Transaction Server, the installation process sets PROPATH for you in the ubroker.properties file. 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 modify this and other environment variable settings in the OpenEdge shell scripts installed on UNIX. 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 environment variable setting, stored in the WRKDIR environment variable, 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.

Configuring WebSpeed and NameServer log files

For disk management reasons, you might want to specify a nondefault 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 log files to reside, you can specify the location for each in OpenEdge Explorer or by directly editing the ubroker.properties file. For more information, see the “Configuring WebSpeed components” section on page 10–19.

Log 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 file with a text editor. You might want to archive the file contents as you do it. For more information on maintaining log files, see the “Maintaining the WebSpeed Transaction Server and NameServer log files” section on page 10–22.
Configuring WebSpeed components

You can use OpenEdge Explorer to configure the NameServer and the WebSpeed Transaction Server. Alternately, you can use the command-line utilities to manage the configuration.

For more information about creating, configuring, and managing with OpenEdge Explorer, see the online help.

For information about managing with the command-line utilities, see the “WebSpeed command-line utilities” section on page 10–4.

Load balancing with the NameServer

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 your product includes load balancing (for example, the WebSpeed Enterprise Transaction Server) or you have installed the load-balancing option for a product that requires it (for example, the AppServer), 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 Explorer or by editing the priorityWeight property in the ubroker.properties file.

For more information on load balancing and fault tolerance, see the “Understanding the NameServer’s load-balancing option” section on page 9–25, and OpenEdge Getting Started: Installation and Configuration.

Note: The NameServer can be on any machine in your network, even in Windows. In addition, you can configure your Transaction Server to run without a NameServer. For more information, see the “Eliminating the NameServer” section on page 9–23.
Starting the WebSpeed Transaction Server and NameServer

The WebSpeed installation includes one sample NameServer and one sample WebSpeed Transaction Server instance.

To begin to develop or deploy applications on UNIX:

1. The AdminServer must be running. If the AdminServer is not running, use the PROADSV command to start it. (See the “Starting the AdminServer” section on page 10–16.)

2. Start an existing NameServer or create a new NameServer instance. You can create and start a NameServer by using 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 AdminServer starts.

   If you are using OpenEdge Explorer, see the online help for information about creating and starting an instance. If you are editing the ubroker.properties file, see the “Editing the ubroker.properties file” section on page 10–13.

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

   ```
   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:

   ```
   nsman -name NS1 -host host-name -port port -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 AdminServer. The user-name and password in the properties file are used when starting a NameServer in this manner from UNIX.

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 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 AdminServer starts.

   If you are using OpenEdge Explorer, see the online help for information about creating and starting an instance. If you are editing the ubroker.properties file, see the “Editing the ubroker.properties file” section on page 10–13.

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

   ```
   wtbman -name wsbroker1 -start
   ```
Where wsbroker1 is the name of the WebSpeed Transaction Server.

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

```
wtbman -name wsbroker1 -host host-name -port port -start
```

Where `host-name` is the name of the host machine where you want the instance to run; `port` is the port number on the AdminServer. The user-name and password in the properties file are used when starting a NameServer in this manner from UNIX.

By using either OpenEdge Explorer 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 Explorer online help. Also see the “Using the OpenEdge Explorer to check status” section on page 10–25 and the “Managing the WebSpeed Transaction Server” section on page 10–27.
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. Thus, the information in these log files can be useful when you perform routine maintenance or troubleshoot.

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 how to configure the log files for your environment, see the “Configuring WebSpeed and NameServer log files” section on page 10–17.
Configuring a WebSpeed 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. Instead, you can install the Web server and the WebSpeed Messengers together on a different machine if you want.

In this configuration, the Messenger must be able to connect remotely to the machines where the NameServers and AdminServers are installed. To do this, you must configure a remote NameServer. For more information on this, see OpenEdge Getting Started: Installation and Configuration.
Starting WebSpeed to test the configuration

After you set up the Transaction Server machine and the Web server machine, you are ready to run WebSpeed and test your configuration.

To run WebSpeed and test your configuration:

1. Make sure the AdminServer is running. If the AdminServer is not running, use the PROADSV utility, which is installed in install-path/bin, as shown:

   ```
   proadsv -start
   ```

2. Make sure the NameServer is running. If the NameServer is not running, use the NSMAN utility to start it.

   Typically, the sample NameServer (NS1) starts automatically with the AdminServer. You can use this sample NameServer, or you can create a new NameServer by editing the ubroker.properties file. Start the new NameServer with the NSMAN utility.

3. Start an existing WebSpeed Transaction Server. You can use the sample WebSpeed Transaction Server (wsbroker1), or you can create a new Transaction Server by editing the ubroker.properties file; then start the new Transaction Server with the WTBMAN command.

   You can use OpenEdge Explorer to start the NameServer or the WebSpeed Transaction Server. See the OpenEdge Explorer online help for information.
Testing your configuration

To test your configuration:

- Use OpenEdge Explorer to view a detailed status
- Use the -query option of the NSMAN, PROADSV, and WTBMAN utilities to check component status
- View the Messenger Administration page (WSMAdmin)
- Run the status.p procedure from a Web browser

Using the OpenEdge Explorer to check status

You can check the WebSpeed configuration status from OpenEdge Explorer. Select the broker and choose Status. See the OpenEdge Explorer online help for more information.

Using the query option to check status

You can test your configuration and check its status by running the -query option on the AdminServer, NameServer, and the WebSpeed Transaction Server.

To query the AdminServer, use the PROADSV utility, which is installed in install-path/bin, as shown:

```
proadsv -query
```

To query a local NameServer, use the NSMAN utility, which is installed in install-path/bin, as shown:

```
nsman -name NameServer-name -query
```

To query a remote NameServer, add the following parameters to the command:

```
nsman -name NameServer-name -host host-name -port port -user user-name -query
```

To query a local WebSpeed Transaction Server, use the WTBMAN utility, which is installed in install-path/bin, as shown:

```
wtbman -name ts-name -query
```
Where ts-name is the name of the Transaction Server that you want to query. You can have more than one Transaction Server running on a single machine.

To query a remote WebSpeed Transaction Server, add the following parameters to the command:

```
wtbman -name ts-name -host host-name -port port -user user-name -query
```

The query reports on the Transaction Server’s pool of WebSpeed agents. For each agent, it lists a process ID, its port number, its status, how many requests it has serviced, when it started, and when its status changed. The following are the agent status types:

- **AVAILABLE** — The agent is available to execute application logic.
- **BUSY** — The agent is actively executing application logic (Web object) for a Web user.
- **LIMBO** — The agent is in a transitional state. If the status persists, it indicates an error condition.
- **LOCKED** — The agent is dedicated to a particular Web browser and is only available to the browser whose application locks it.
- **STARTING** — The broker has launched the agent, but the agent has not yet initialized.

**Viewing the Messenger Administration page**

You can test to see if the Web server is fully operational by pointing your Web browser to the WSMAdmin page and following the directions. The WSMAdmin page allows you to perform a round-trip test of your entire WebSpeed configuration or to test each component or leg separately.

See the “Managing the WebSpeed Messenger” section on page 10–29 for the URL for the WSMAdmin page.

**Running the status.p procedure**

You can run the status.p procedure from a Web browser. The status.p procedure is installed into the `install-dir/src/web/examples` directory. Running this procedure lets you check that WebSpeed and your Web server are fully operational. If you successfully run this procedure, it means that you configured the WebSpeed run-time environment correctly. If you then have errors running your own application, you can be confident that the problem is specific to your application files.

To run the status.p procedure from a Web browser, enter a URL using the following format:

```
http://hostname/script-dir/Messenger/WService=brokername/src/web/examples/status.p
```
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 the Transaction Server with the WSCONFIG utility. The validation reads the file and reports the status of the Transaction Server. The WTBMAN utility commands run remotely; the WSCONFIG utility commands only run locally.

You can also use OpenEdge Explorer to manage and validate the Transaction Server. (With OpenEdge Explorer, you can also configure the Transaction Server.) See the OpenEdge Explorer online help for more information.

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

- 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 Transaction Server 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 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 Transaction Server

To stop the Transaction Server 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
```
Managing the WebSpeed Messenger

Sometimes you might want to check the Messenger's status. For example, the NSAPI Messenger maintains persistent connections to the WebSpeed Transaction Servers, 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 where 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 Transaction Server or NameServer.

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 a CGI Web server, use the following URL:

```
http://host-name/scripts/cgiip.exe[/WService=appservice-name]?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.
This chapter explains how to configure WebSpeed dynamic code-page support.

To understand this chapter, you should be an experienced WebSpeed developer familiar with internationalization, code pages, and Unicode. For information on developing WebSpeed applications, see *OpenEdge Application Server: Developing WebSpeed Applications*. For information on internationalization, code page, and Unicode, see *OpenEdge Development: Internationalizing Applications*.

This chapter contains the following sections:

- Introduction
- Basic browser-agent interaction
- How dynamic code-page support works
- Summary of code-page conversions
- Configuring the agent
- Tasks for the WebSpeed developer
WebSpeed dynamic code-page support lets a WebSpeed agent running one code page and a Web browser running a different, but compatible, code page exchange data without corrupting it.

With dynamic code-page support, a single agent with the CPINTERNAL code page set to UTF-8 (an encoding of Unicode) can handle requests from multiple browsers each running a different code page.
Basic browser-agent interaction

In order to understand dynamic code-page support, it is necessary to understand how the Web browser and the WebSpeed agent interact in a typical WebSpeed configuration. (For information on typical WebSpeed configurations, see OpenEdge Getting Started: WebSpeed Essentials.) For the moment, ignore the Web server, the WebSpeed Messenger, the WebSpeed broker, and code pages. Assume that a browser and an agent have established communication.

The browser and agent interact as follows:

1. The browser sends a Web request.
2. The Web request is received by the agent.
3. The agent runs the Web Object (the WebSpeed application).
4. The WebSpeed application creates a Web page.
5. The agent sends the Web page.
6. The Web page is received by the browser.

The preceding steps repeat.
How dynamic code-page support works

In dynamic code-page support, each Web request sent by the browser indicates, directly or indirectly, the name of the code page it is encoded in. The developer modifies the application to make this happen. Typically, the initial Web page of the application displays a preselected list of languages or nationalities, from which the user selects.

When a Web request reflecting a response to a language-selection page is received by the agent, the application does the following:

1. Notes the language or nationality selected
2. Determines the corresponding code page
3. Determines the MIME version (as opposed to the OpenEdge version) of the code-page name
4. Converts the MIME version of the code-page name to the OpenEdge version
5. Sets a cookie called \texttt{wscharset} to the MIME version of the code-page name

Or:

Sets a query field or a hidden-form field called \texttt{wscharset} to the MIME version of the code-page name

When a Web request reflecting a response to a Web page other than a language-selection page is received by the agent, WebSpeed’s \texttt{web-disp.p} procedure:

1. Searches for the code-page name in a query field called \texttt{wscharset} or a hidden-form field called \texttt{wscharset}. If the code-page name was not found, it searches in a cookie called \texttt{wscharset}.
2. Converts the code-page name from MIME format to OpenEdge format and stores the result in the \texttt{WEB-CONTEXT} handle’s \texttt{HTML-CHARSET} attribute.

Then, the application ensures that the MIME version of the code-page name will appear in the Web page the application will create. The technique used depends on whether the Web page is created using static HTML or dynamic HTML. For example:

- If the Web page is created using static HTML, the application should contain a \texttt{META} tag with the MIME version of the code-page name, as in the following:

  ```
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset="windows-1250"">
  ```

- If the Web page is created using dynamic HTML, the application uses the \texttt{OUTPUT-CONTENT-TYPE()} WebSpeed API function to insert the MIME version of the code-page name into the HTML document.
Once WebSpeed’s `web-disp.p` procedure stores the OpenEdge version of the code-page name in `HTML-CHARSET`, the agent uses `HTML-CHARSET` to perform the following code-page conversions:

1. Just after an incoming Web request is received, the agent converts it from the `HTML-CHARSET` code page to the agent’s `CPINTERNAL` code page.

2. Just before an outgoing Web page is sent, the agent converts it from the agent’s `CPINTERNAL` code page to the `HTML-CHARSET` code page.
Summary of code-page conversions

With dynamic code-page support:

- When a WebSpeed agent receives a Web request, if the `WEB-CONTEXT` handle’s `HTML-CHARSET` attribute corresponds to the OpenEdge version of a code-page name, the agent converts the Web request from the `HTML-CHARSET` code page to the agent’s `CPINTERNAL` code page.

- Just before the agent sends a Web page, if the `WEB-CONTEXT` handle’s `HTML-CHARSET` attribute is set, the agent converts the Web page from the agent’s `CPINTERNAL` code page to the `HTML-CHARSET` code page.

Without dynamic code-page support:

- When an agent receives a Web request, it is not converted.

- Just before the agent sends a Web page, the agent converts it from the agent’s `CPINTERNAL` code page to the `SESSION:CPSTREAM` code page.
Configuring the agent

The CPINTERNAL code page of a WebSpeed agent must be compatible with the code page of any Web request it receives. To accomplish this easily, make the agent’s CPINTERNAL code page UTF-8, which is compatible with virtually all code pages OpenEdge supports.

For more information, see Chapter 12, “Connecting WebSpeed to a Data Source.”
Tasks for the WebSpeed developer

For dynamic code-page support to work, the WebSpeed developer must perform certain preliminary tasks. To summarize dynamic code-page support from the perspective of the WebSpeed developer:

1. The WebSpeed application should include a page that lets the user select a language or nationality from a list of preselected languages or nationalities. For each language or nationality in the list, the developer determines an appropriate code page, which must be compatible with the WebSpeed agent’s CPINTERNAL code page. To simplify meeting this requirement, make the agent’s CPINTERNAL code page UTF-8, which is highly compatible with OpenEdge code pages. (For more information, see install-path/prolang/readme.)

2. The user starts the application and selects a language or nationality.

3. The application notes the language or nationality selected, determines the corresponding code page, and stores the MIME version of the code-page name within the Web request in a query field, a hidden-form field, or a cookie.

Note: Any static HTML the application uses should be in the same code page selected by the user and should refer to the same code page in the <META> tag’s HTTP-EQUIV attribute. See the first code example in the “Storing the MIME code-page name” section on page 11–10.

4. Dynamic code-page support takes over.

Each time the agent receives information from a Web browser, it automatically converts the information from the browser’s code page to the agent’s CPINTERNAL code page. Similarly, just before the agent sends a Web page back to the browser, it automatically converts it from the agent’s CPINTERNAL code page to the browser’s code page.
Determining the MIME code-page name

The application must store the MIME code-page name, not the OpenEdge code-page name. Table 11–1 gives the MIME name and the OpenEdge name for several common code pages.

Table 11–1: MIME and OpenEdge code-page equivalents

<table>
<thead>
<tr>
<th>MIME name</th>
<th>OpenEdge name</th>
</tr>
</thead>
<tbody>
<tr>
<td>iso-8859-2</td>
<td>iso8859-2</td>
</tr>
<tr>
<td>Shift_JIS</td>
<td>SHIFT-JIS</td>
</tr>
<tr>
<td>TIS-620</td>
<td>620-2533</td>
</tr>
<tr>
<td>windows-1250</td>
<td>1250</td>
</tr>
</tbody>
</table>

Note: Table 11–1 does not list all code pages that can be used with dynamic code-page support. For a complete list of these code pages, see the source code for adecomm/convcp.p.

For more information on converting between MIME and OpenEdge code-page names, see the “Converting code-page names between MIME and OpenEdge formats” section on page 11–11.
Storing the MIME code-page name

To store the MIME code-page name, use one of the following techniques:

- Using HTML, store the MIME code-page name in a hidden-form field or a query field called wscharset.

  **Note:** The field must be called wscharset.

In the following fragment, the MIME code-page name is stored in a hidden-form field called wscharset:

```html
<HEAD>
  <META HTTP-EQUIV="Content-Type"
       CONTENT="text/html; charset=windows-1252">
</HEAD>

<FORM METHOD="POST" ACTION="test.w">
  Name: <INPUT TYPE="text" ID="name"><BR>
  Address: <INPUT TYPE="text" ID="address"><BR>
  City: <INPUT TYPE="text" ID="city"><BR>
  <INPUT TYPE="hidden" NAME="wscharset" VALUE="windows-1252">
</FORM>
</BODY>
```

In the following fragment, the MIME code page is stored in a query field called wscharset:

```html
<A HREF="tstiso2.w?wscharset=iso-8859-2"></A>
```

- Use the `SET-COOKIE` WebSpeed API function to create a cookie called wscharset containing the MIME code-page name, as shown in the following fragment:

  ```javascript
  ```

- Dynamic code-page support does not work if the code-page name is stored using the `CHARSET` attribute of the `META` tag exclusively or the WebSpeed API `outputContentType()` function exclusively. You must also store the code-page name in wscharset. Also, the stored code-page names must match.

Accessing WEB-CONTEXT’s HTML-CHARSET attribute

The WebSpeed agent stores the OpenEdge version (as opposed to the MIME version) of the code-page name in `WEB-CONTEXT:HTML-CHARSET` attribute. Once `HTML-CHARSET` is assigned this value, it can be accessed by the WebSpeed application.

**Note:** The WebSpeed application should not modify `WEB-CONTEXT:HTML-CHARSET`. 
Converting code-page names between MIME and OpenEdge formats

You can convert a code-page name between MIME and OpenEdge formats by running the convcp.p procedure, which resides in the adecomm directory.

This is the convcp.p syntax:

**Syntax**

```plaintext
convcp.p(input-name, direction:U, OUTPUT output-name)
```

**input-name**

A CHARACTER expression indicating the code-page name to be converted.

**direction**

A CHARACTER indicating one of the following:

- “toProg” (to convert from MIME to OpenEdge).
- “toMime” (to convert from OpenEdge to MIME).

**output-name**

A CHARACTER expression indicating the name of a CHARACTER variable to contain the result.

If `input-name` is not a valid MIME or OpenEdge code-page name, `output-name` is set to blank.

The following example converts the value of `cMimeCharset` to OpenEdge format and stores the result in `cProCharset`:

```plaintext
RUN adecomm/convcp.p(cMimeCharset, “toProg”:U, OUTPUT cProCharset).
```

The following example converts the value of `cProCharset` to MIME format and stores the result in `cMimeCharset`:

```plaintext
```
Additional notes

The following are additional notes on developing applications with dynamic code-page support:

- The page for selecting a language or nationality not use fill-in fields. If fill-in fields are used, the application might not be able to determine which code page the fill-in fields are encoded in.

- If your Web server can perform code-page conversion, turn this feature off if you want to use the dynamic code-page support supplied by WebSpeed.

- See the “Creating an international Web site” section on page 12–8 for information on how to configure WebSpeed to support internationalization.
Connecting WebSpeed to a Data Source

This chapter describes how to connect WebSpeed agents to a data source for application development or deployment, as outlined in the following sections:

- Connecting to an OpenEdge RDBMS
- Connecting to a non-OpenEdge data source through a DataServer
- Creating an international Web site
Connecting to an OpenEdge RDBMS

WebSpeed supplies a demonstration OpenEdge database called Sports2000. You can use the Sports2000 database to run the sample applications and experiment with your first WebSpeed applications.

In a local configuration, the OpenEdge database resides on the same machine as the WebSpeed Transaction Server. The structure of a remote configuration can vary. OpenEdge Getting Started: WebSpeed Essentials shows some of the configuration possibilities. When you are developing in a client/server environment, you can connect to the database in single-user mode. When you develop WebSpeed applications in Windows with the WebSpeed Workshop, you must run the OpenEdge database in a multi-user mode. In the WebSpeed Workshop development environment, several agents must connect to the database simultaneously: one for OpenEdge Explorer, another for the WebSpeed WebTools, and one for the AppBuilder.

See the chapter on startup in OpenEdge Data Management: Database Administration for more information on running the OpenEdge database server and the parameters that you can specify.

You provide the information that an agent needs to connect to a database when you configure a WebSpeed Transaction Server and specify the agent options for its pool of agents. The WebSpeed agent is an ABL client running in batch mode and can accept nearly all the startup parameters a standard client can.

The following is a sample `srvrStartupParam` setting in `ubroker.properties` for the sample Transaction Server, `wsbroker1`, running in Windows:

```
    srvrStartupParam=-p web\objects\web-disp.p -cpstream iso8859-1 -weblogerror
```

**Note:** The `srvrStartupParam` property is a dynamic property. Any changes to this property only affects new agents started after the value changes because the change affects how the agent is started. Existing (running) brokers and agents are not updated with the new value.

WebSpeed provides many parameters that allow you to control connections to the database and make adjustments to increase performance. These include parameters for controlling record-buffer size, cursor size, and read-only access. In addition to database connection parameters, WebSpeed provides session parameters that allow you to control the conditions under which it compiles your r-code and Web objects and how to handle code page issues. See OpenEdge Deployment: Startup Command and Parameter Reference for details on session parameters.

Connecting programmatically

You can also provide connection information programmatically with the SpeedScript `CONNECT` statement. You can specify a parameter file that includes connection information with the `CONNECT` statement. However, you cannot connect to a database in the same procedure that accesses that database. You cannot connect to a database within an embedded SpeedScript program that then accesses the database tables and fields. Instead, you should include the `CONNECT` statement in a separate procedure (.p) that the embedded SpeedScript calls. This ensures that all agents receive the information they need to connect to the database.
Connecting to a local database

The parameter file that contains information for connecting to a local database must include at least the database name. If the database files do not reside in the agent’s working directory, you must fully qualified the database pathname. If you want the agent to connect to more than one database at a time, you must use the full database path with the Physical Database Name (-db) parameter to specify each additional database. The following parameter file connects an agent to the Sports2000 demonstration database:

```
/srvrStartupParam=-p web\objects\web-disp.p -cpstream iso8859-1 -weblogerror -db c:\wrk\sports2000
```

Note: The `srvrStartupParam` property is a dynamic property. Any changes to this property only affects new agents started after the value changes because the change affects how the agent is started. Existing (running) brokers and agents are not updated with the new value.

By default, this connection uses shared memory. If you want to use a network connection instead, specify the Host Name (-H) and Service Name (-S) parameters (described in Table 12–1) when you connect to the database. The network type defaults to TCP.

Connecting to a remote database

When connecting to a remote database, an agent needs information about the database’s physical location. Unlike a local connection, where you can choose not to include them, you must include the parameters described in Table 12–1 for a remote connection.

Table 12–1: Remote connection parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Name (-H)</td>
<td>Indicates the name of the machine where the database server resides.</td>
</tr>
<tr>
<td>Service Name (-S)</td>
<td>Indicates the name of the available service you are calling. This is the service that the database server uses, not a WebSpeed service. See your <code>\etc\services</code> file for a list of available services. Note that the database server does not have to use a named service.</td>
</tr>
</tbody>
</table>
The following example of a `SrvrStartupParam` setting from `ubroker.properties` connects to a remote Sports2000 database:

```
SrvrStartupParam=-p web\objects\web-disp.p -cpstream iso8859-1 -weblogerror
-db sports2000 -H pluto -S mysv12
```

**Note:** The `SrvrStartupParam` property is a dynamic property. Any changes to this property only affects new agents started after the value changes because the change affects how the agent is started. Existing (running) brokers and agents are not updated with the new value.

The remote database is on the `pluto` host and the database server uses the service, `mysv12`.
Connecting to a non-OpenEdge data source through a DataServer

A WebSpeed application can access a range of databases through the OpenEdge DataServer architecture. There are native DataServers for accessing the ORACLE DBMS and Microsoft SQL Server. The OpenEdge DataServer for ODBC allows applications to access Informix OnLine, Microsoft Access, SYBASE, and DB2.

The following sections supplement the information provided by the applicable OpenEdge DataServer guide. The sections describe the general tasks that you must perform to install and configure your DataServers in a WebSpeed environment. The sections also provide specific instructions for connecting WebSpeed agents to the various data sources.

Before developing a WebSpeed application that accesses a non-OpenEdge data source, read your DataServer Guide for information on writing SpeedScript code that gives you the results and behavior you expect for your data source. Some elements of SpeedScript do not behave as documented in the standard WebSpeed documentation when accessing non-OpenEdge data sources. There is also additional syntax that allows you to optimize queries by leveraging the strengths of individual database managers that the standard documentation does not describe in detail.

Typically, integrating a non-OpenEdge data source into your WebSpeed environment involves following these basic steps:

- Planning where the DataServer fits in your WebSpeed hardware architecture
- Installing and setting up the DataServer
- Building a schema holder that contains the data definitions in a format required by WebSpeed applications
- Configuring WebSpeed agents to connect to the schema holder and data source

Planning where to locate a DataServer is a similar process regardless of which DataServer you use. The other steps, however, vary across DataServers. See the appropriate OpenEdge DataServer guide for instructions.
DataServer and WebSpeed architecture

You have the option of having a local or a remote DataServer. A local DataServer runs on the same machine as the WebSpeed agents. A remote DataServer runs on a different machine from the WebSpeed agents.

Figure 12–1 shows a DataServer that is local to the WebSpeed Transaction Server. The database also resides on this machine. (However, the database does not have to be local.)

Figure 12–1: The local DataServer

Figure 12–2 shows one possible configuration for the remote DataServer where a WebSpeed agent accesses a remote DataServer. Here, the database and the DataServer are running on the same machine. The schema holder is local to the WebSpeed Transaction Server to increase performance.

Figure 12–2: Remote DataServer

The figures show a schema holder on the same machine where your agents run. A local read-only schema holder gives you better performance. However, you might decide that you want your schema holder on a separate machine to make better use of processing resources.

Note that none of the configuration diagrams include a Web server. The Web server plays no role after passing the initial Web request for data to the WebSpeed Messenger. Much like a typical OpenEdge client, the WebSpeed agent interacts with the DataServer, which in turn passes queries and data requests to the database management system.
You can integrate DataServers for ODBC and ORACLE into the unified broker framework, which allows you to have a single administrative system and take advantage of the NameServer’s load balancing. Alternately, the WebSpeed agents can access DataServers that cannot be managed by the AdminServer.

There are variations on these configurations that allow you to combine Windows and UNIX platforms or to distribute the components using vendor networking (for example, ORACLE’s SQL*Net). See your OpenEdge DataServer guide for more information. The DataServer guides present configuration information in traditional client/server terms; when reading this material for implementing WebSpeed, substitute WebSpeed agent for Client.

## Setting up and running the DataServer

After installing and configuring your DataServer as described in an OpenEdge DataServer guide, you can set up and run a DataServer for WebSpeed.

### To set up and run a DataServer for WebSpeed:

1. Start the DataServer processes and create the schema holder. The WebSpeed Transaction Server does not have to be running at this time.

2. One recommended technique for managing an agent’s connection information is to create a parameter file (.pf). Create a parameter file that specifies the schema holder, the database, user information, and information for accessing a remote database, if your configuration requires it. For more information on creating and using parameter files, see OpenEdge Deployment: Managing ABL Applications.

3. Configure the WebSpeed agent to use the parameter file that you created by specifying the Parameter File (-pf) parameter. For example, if you created a parameter file named `db2dbcon.pf`, your entry in the Startup Parameters field on the Server Parameters tab in OpenEdge Explorer might be:

   ```
   -p web\objects\web-disp.p -cpstream iso8859-1 -pf db2dbcon.pf
   ```

4. Start the Transaction Server.

   When the WebSpeed broker whose agents you configured with DataServer connection information receives a Web request, it selects an agent from its pool to service the request by connecting to the non-OpenEdge database and, through the DataServer, retrieving or updating data.
Creating an international Web site

If your Web site targets an international audience, consider letting users select the language or nationality they prefer. For each language or nationality, besides translating the text, you can adapt graphics, images, layout, and content. For example, if you are building a Portuguese-language Web site that deals with currency and taxes and that targets users in Portugal and in Brazil, you can adapt the content to deal with Portuguese currency and taxes as well as with Brazilian currency and taxes.

Also, consider using UTF-8 (an encoding of Unicode) for the encoding of the database and for the CPINTERNAL code page of the WebSpeed agents. This approach is strongly recommended by Progress Software Corporation. For more information on Unicode, see *OpenEdge Development: Internationalizing Applications*.

Also, consider using WebSpeed dynamic code-page support. For more information, see Chapter 11, “WebSpeed Dynamic Code-page Support.”

Figure 12–3 shows a WebSpeed configuration that supports an international Web site.

![WebSpeed configuration that supports an international Web site](image)
This configuration creates a Web site that handles requests from users in Spain, Japan, and the United States. In this example, all countries share a single Unicode (UTF-8) database. This particular Web site expects the highest rate of access from its Spanish users, so it has multiple Transaction Servers and agent pools configured to handle the Web requests. These Transaction Servers are, in turn, registered with a NameServer. The NameServer can manage the Spanish Transaction Servers as they have access to equivalent resources, that is, their agents are all configured identically: they use the same internal code page and they connect to the same database in the same manner. Web requests coming from Japanese and U.S. users are each serviced by a single Transaction Server and its agent pool.

**Setting up Transaction Servers**

To set up a Transaction Server to support part of an international WebSpeed application, set up the environment in which it runs and configure its agent pool to access the appropriate data. To simplify configuration and maintenance, consider creating a subdirectory for each language or nationality. Similarly, consider placing databases, parameter files, and other resources in directories specific to the language or nationality.

When setting up a Transaction Server’s environment, you must ensure that when a message is issued by a WebSpeed agent, the Transaction Server displays it in the appropriate language. To accomplish this, set the PROMSGS variable in the ubroker.properties file’s Environment section for each Transaction Server definition. You cannot set PROMSGS at a system level, because multiple Transaction Servers requiring different translations of messages will run simultaneously. The international PROMSGS files have a three-letter extension that indicates the language. The following example sets PROMSGS to the Japanese version of the messages file:

```
PROMSGS=/usr/wsrt/prolang/jpn/promsgs.jpn; export PROMSGS
```

Define and configure your Transaction Servers as you would for a standard WebSpeed installation.

If it is important that you have messages displaying in the user’s language at all levels of the application, you can use the CGI-compatible Messenger. The NSAPI-compatible Messenger can use only one PROMSGS file, which you configure in its startup script. For the CGI Messenger, you can create a script file (based on wspd_cgi.sh) for each language and specify the appropriate PROMSGS file to use.

**Configuring agents**

To configure agents to run in an international environment, you must address code-page issues and session options when you specify agent parameters. OpenEdge automatically installs the language you select as the default and configures your machine for this default language; however, you must configure each agent to use the appropriate parameters for the country it accepts requests from. Using a parameter (.pf) file helps you manage agent startup and connection information.

In Windows, use OpenEdge Explorer to configure the agents. You add the Parameter File (-pf) startup parameter and name of your parameter file to the Startup Parameters field for the agent Executable File.
On UNIX, use OpenEdge Explorer or manually update the ubroker.properties file to configure the agents. You add the Parameter File (-pf) startup parameter and name of your parameter file to the srvrStartupParam option in the section of the ubroker.properties file where you define the Transaction Server.

**Note:** The srvrStartupParam property is a dynamic property. Any changes to this property only affects new agents started after the value changes because the change affects how the agent is started. Existing (running) brokers and agents are not updated with the new value.

For each supported international language, OpenEdge provides an example parameter file that you can use as the base for creating your own.

The following examples show the contents of parameter files for the Spanish, Japanese, and U.S. agents, respectively. They are based on the parameter files provided by OpenEdge in the prolang/lang directory, as shown:

```plaintext
-db intldb -cpinternal SHIFT-JIS -cpstream SHIFT-JIS -d ymd

-db intldb -cpinternal iso8859-1 -cpstream iso8859-1 -d dmy -E

-db intldb -cpinternal iso8859-1 -cpstream iso8859-1
```

These files provide the information that agents need to connect to a local database and to run with session options appropriate to their cultural context. For example, each agent displays dates in a distinct format; in addition, the Spanish agents use the comma as a radix for decimal data. You can add other parameters to the parameter file to further manage how agents connect to the databases.

The following is a sample setting for the srvrStartupParam option in the section of the ubroker.properties file where you define the wsJapan Transaction Server:

```plaintext
srvrStartupParam=-p web/objects/web-disp.p -cpstream iso8859-1 -weblogerror -p web/objects/web-disp.p -weblogerror -pf japan.pf
```

**Note:** The srvrStartupParam property is a dynamic property. Any changes to this property only affects new agents started after the value changes because the change affects how the agent is started. Existing (running) brokers and agents are not updated with the new value.

Alternatively, the WebSpeed application can let the language or nationality be selected by the user. From the user’s selection, the application can set the CPINTERNAL, CPSTREAM, and DATE-FORMAT attributes of the SESSION handle to the appropriate values.
Creating an international Web site

Code pages

The agent’s CPINTERNAL code page (that is, the code page used by the agent for its internal processing) must be compatible with the code page of the database it connects to. This is easy to accomplish if the databases’s code page is UTF-8.

The agent’s CPSTREAM code page (which the -cpstream parameter specifies) must be the same as the agent’s CPSTREAM code page (assuming dynamic code-page support is not used).

Session options

Supporting an international Web site means more than just supporting several languages. You must also support the cultural difference in date formats and numeric notations. WebSpeed allows you to control how agents display and process date information and numeric conventions. The Date (-d) parameter lets you specify the format that an agent uses to process dates. By default, an agent processes dates as month, day, year. To display a date as day, month, year, specify -d dmy in the agent’s parameter (.pf) file. The European Numeric Format (-E) parameter specifies that a comma (,) represents the decimal point instead of a period (.)

In Windows, another aspect of an agent’s session environment that you control through startup parameters is which initialization (.ini) file an agent uses. If you want an agent to display error messages in a language other than the default language you chose for your WebSpeed installation, you must use an initialization file to set the PROMSGS option to the appropriate version of the PROMSGS file. The WebSpeed agent can use the [WinChar Startup] section of the initialization file, so you must set PROMSGS there. Specify the Initialization File (-ininame) parameter to specify the name of the initialization file you customized for the agent.

Directing Web requests

As you design your international Web site, keep in mind that one of the initial pages should let users choose a specific language or nationality. The URL for each language option should specify the appropriate Transaction Server for the language the user has chosen.

Assuming that the sample configuration shown in Figure 12–3 is running with an NSAPI-compatible Web server, the URLs for the links that Japanese, Spanish, and U.S. users would select are as follows:

http://mars/wsnsa.dll/WService=wsJapan/iwebapp

http://mars/wsnsa.dll/WService=wsUSA/iwebapp

http://mars/wsnsa.dll/WService=wsSpain/iwebapp

The URLs for Japan, Spain, and the U.S. name WebSpeed Transaction Servers to service the Web request.
This chapter highlights some security issues related to WebSpeed, as described in the following sections:

- Changing WebSpeed applications from development mode to production mode
- Changing additional settings to minimize security risks
- Authenticating a password using SpeedScript
- Securing data transmissions between WebSpeed client and server components
- Maximizing WebSpeed compatibility with your firewall
Changing WebSpeed applications from development mode to production mode

Changing your WebSpeed applications from development mode to production mode allows you to minimize some basic system vulnerabilities.

To transition from development mode to production mode, you must:

- Set the agent application mode to production
- Disable the WebSpeed Messenger Administration utility
- Establish the WebSpeed Messenger Administration Internet Protocol List (Optional)
- Check the status of the Debug mode

You can access these properties from either OpenEdge Explorer, or the ubroker.properties file that resides in the install-path\properties directory. You can use the mergeprop utility installed with OpenEdge to manually edit the ubroker.properties file. For information on using mergeprop, see OpenEdge Getting Started: Installation and Configuration.

Set the agent application mode to production

You must set the value of your agent application mode to production to prevent unwanted exposure of several WebSpeed components. The default setting is development. Therefore, you must take explicit action to change to production mode before you deploy and run your WebSpeed applications.
Changing WebSpeed applications from development mode to production mode

Table 13–1 details the differences between production and development modes.

Table 13–1: Agent application mode descriptions

<table>
<thead>
<tr>
<th>When the agent application mode is set to . . .</th>
<th>Then this property . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production</td>
<td>• Prohibits Web objects specified in the URL with the path <code>install-path\src\web\</code> from being run</td>
</tr>
<tr>
<td></td>
<td>• Disables these utility commands: debug, reset, and ping</td>
</tr>
<tr>
<td></td>
<td>• Disables access to WebTools</td>
</tr>
<tr>
<td></td>
<td>• Restricts the utility command ping so that it only indicates that the WebSpeed agent is available; it does not display any other information about the agent’s environment</td>
</tr>
<tr>
<td></td>
<td>• Ignores debugging via the URL, <code>?debug=xxx</code>, even if the Debug mode is enabled</td>
</tr>
<tr>
<td>Development (default)</td>
<td>• Allows unlimited access to run Web objects specified in the URL with the path <code>install-path\src\web\</code></td>
</tr>
<tr>
<td></td>
<td>• Enables these utility commands: debug, reset, and ping</td>
</tr>
<tr>
<td></td>
<td>• Enables access to WebTools</td>
</tr>
<tr>
<td></td>
<td>• Allows users to access specific development files</td>
</tr>
<tr>
<td></td>
<td>• Enables debugging via the URL, <code>?debug=xxx</code>, if Debug mode is enabled</td>
</tr>
</tbody>
</table>

You can use OpenEdge Explorer to change the value of the Agent application mode from development to production.

To change the application mode with OpenEdge Explorer:

1. Select the WebSpeed folder on the left-hand side of the main OpenEdge Explorer window.
2. Click Configuration and then Edit.
3. Select the Agent tab. Note that the Server application mode field displays two options: Development and Production.
4. Set the Agent application mode to Production.
Another method of changing the application mode from development to production is to edit the `ubroker.properties` file.

**To change the application mode by editing `ubroker.properties`:**

1. Determine if you need to set all brokers or just individual brokers to a production mode.
   
The property values established in the various sections of the `ubroker.properties` file are governed by a parent-to-child inheritance relationship. For example, values set in the parent, the `[UBroker]` section, are inherited at the `[UBroker.WS]` level. Similarly, values defined at the `[UBroker.WS]` level are inherited as default values by each individual broker instance in the `[UBroker.WS.brokername]` sections.
   
   Alternatively, you can retain the property values established at the parent level and override only those individual values at the lower, child level that you need to change.

2. Search the `ubroker.properties` file for the `srvrAppMode` property in the section, or sections, that you need to change. For example:
   
   - To affect all brokers, search for `srvrAppMode` property in the `[UBroker.WS]` section.
   
   - To affect one or more individual brokers, search for or add the `srvrAppMode` property in the individual `[UBroker.WS.brokername]` section.

3. Change the value of the `srvrAppMode` property to Production.

**Disable the WebSpeed Messenger Administration utility**

You should prevent access to the WebSpeed Messenger Administration (WSMAadmin) utility when you deploy your WebSpeed applications. It provides information about the NameServer, WebSpeed broker, WebSpeed agent, and other components of your system. It can also allow users to access static HTML files.

When enabled in the Development mode, this utility can help you debug problems in a WebSpeed configuration. However, if you leave this utility enabled when you move your system from Development mode to Production mode, you expose your system information to unauthorized access.

Ensure that you disable the WebSpeed Messenger Administration utility before you deploy your WebSpeed applications.

There is one exception to this guideline. If you are using WSMAadmin with the WebSpeed Messenger Administration Internet Protocol List option, you can leave the WebSpeed Messenger Administration utility enabled in Production mode. See the “Establish the WebSpeed Messenger Administration Internet Protocol List (Optional)” section on page 13–5 for more information.
Disabling WSMAdmin

You can use OpenEdge Explorer to disable the WSMAdmin utility.

To disable the WSMAdmin utility with OpenEdge Explorer:

1. Double click the Messengers folder on the left side of OpenEdge Explorer.
2. Select the specific messenger whose internal commands you want to disable.
3. Choose Configuration and click Edit.
4. On the Advanced tab, deselect the check box next to the Internal administration command.
5. If you are changing this value for either a WSISA or WSNSA Messenger, re-start your web server.

Verify that the Messenger Internal Commands are disabled. You should not be able to access the WSMAdmin utility and its features from a Web browser.

You can also disable the internal commands for a WebSpeed messenger by editing the ubroker.properties file.

To change disable the WSMAdmin utility by editing ubroker.properties:

1. Search the file for the AllowsMsngrCmds property. It is located in the properties section that is associated with the specific messenger type that your configuration is using.

   For example, if you are using a CGIIP Messenger, the AllowsMsngrCmds property you must set is in the [WebSpeed.Messengers.CGIIP] section.

   **Note:** Do change the AllowsMsngrCmds property in the [WebSpeed.Messengers] section. Changes in this section will not affect the AllowsMsngrCmds property for individual messengers.

2. Set the property value to 0 to disable the WebSpeed Messenger Administration utility commands.

Verify that the Messenger Internal Commands are disabled. You should not be able to access the WSMAdmin utility and its features from a web browser.

Establish the WebSpeed Messenger Administration Internet Protocol List (Optional)

The WebSpeed Messenger Administration Internet Protocol List option allows you to define a list of IP addresses that you can enable to access the WebSpeed Messenger’s internal administrative commands. Any IP address not specified on this list is denied access to WSMAdmin.
You can use the Internet Protocol List option only if you enable the WebSpeed Messenger Administration utility. This utility maintains control over your list of IP addresses. Using the Internet Protocol List allows you the opportunity to debug specific applications remotely without exposing Web applications to unwanted access.

**To add IP addresses using OpenEdge Explorer:**

1. Double click the **Messengers** folder on the left-hand side of OpenEdge Explorer.
2. Select the specific messenger whose internal commands you want to enable.
3. Choose **Configuration** and click **Edit**.
4. On the Advanced tab, check **Internal administration command**.
5. In the **IP address** field, type one or more unique IP addresses.
6. Click the **Save** button.

**To define specific IP addresses using the ubroker.properties file:**

1. Search the file for the `AllowsMsngrCmds` property. It is located in the messenger properties section that is associated with the specific messenger type that your configuration is using.

   For example, if you are using a CGIIP Messenger, you must set or add the `AllowsMsngrCmds` property in the `[WebSpeed.Messengers.CGIIP]` section.

   **Note:** Do change the `AllowsMsngrCmds` property in the `[WebSpeed.Messengers]` section. Changes in this section will not affect the `AllowsMsngrCmds` property for individual messengers.

2. If the `AllowsMsngrCmds` property is disabled, set the property value to `1` to enable it.
3. Add the `wsmAdmIPList` property and type each unique IP address that you want to identify, separating each address with a comma.

   **Caution:** Do not leave the IP address field blank (default value) because it will enable any IP address to access the WebSpeed Messenger’s internal commands.

---

**Check the status of the Debug mode**

The Debug mode allows you to examine and correct the errors that display in the browser when you are running code. This mode can provide information about the WebSpeed agent, the system, and the path of an application.

When your application is in Production mode, the Debug mode is always off, regardless of the value on the Debug setting. When you are in Development mode, you can set values for the Debug mode to enable or disable debugging.
To check the status of the Debug mode using OpenEdge Explorer:

1. Double click the WebSpeed folder on the left-hand side of OpenEdge Explorer.
2. Select the broker name.
3. Choose Configuration and then Edit.
4. Select the Agent tab. The Debug mode field has three options: default, enabled, disabled.
5. The Agent application mode value governs which Debug mode values are available. For example:
   - If the value set for the Agent application mode is Production, the Debug mode is always off, regardless of the value that displayed on the Debug setting.
   - If the value set for the Agent application mode is Development, you can set values for the Debug mode that allow you to enable or disable debugging.

To check the status of Debug mode from the ubroker.properties file, perform:

1. Review the srvrAppMode setting for the broker property sections that you want to affect. It will be either Production or Development.
2. Add the srvrDebug property to the broker property sections that you need to change.
3. Set the value for the srvrDebug mode.
Changing additional settings to minimize security risks

The following information identifies some additional changes you can make when you change your application from development to production mode. These changes further limit your system’s exposure to unauthorized access.

Port numbers and WebSpeed server names

To make it more difficult for personnel outside your organization who know WebSpeed to access your files, it is advisable to change default port numbers and server names, as follows:

- Change all default port numbers to random, available port values between the range of 1025 and 65536 before you deploy WebSpeed. Table 13–2 lists some key default port numbers to consider changing.

Table 13–2: Default port number to change

<table>
<thead>
<tr>
<th>Default port number</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>20931</td>
<td>AdminService</td>
</tr>
<tr>
<td>5162</td>
<td>NameServer</td>
</tr>
<tr>
<td>3050</td>
<td>New WebSpeed brokers</td>
</tr>
<tr>
<td>3055</td>
<td>Default WebSpeed broker (wsbroker1)</td>
</tr>
</tbody>
</table>

- Change all default WebSpeed broker names, AdminService names, and NameServer names from their system-supplied default names to proprietary names to protect their identities.

Minimize the PROPATH entries

Restrict the entries in the PROPATH to application and install-dir/tty file. This technique exposes only the information that is required to run WebSpeed applications.

Do not include references to the following information in the PROPATH:

- FTP directories
- Any file upload directory set in your ubroker.properties file
- Any Web server directories
- OpenEdge-supplied procedure libraries
Techniques to minimize unauthorized access to WebSpeed Messengers

Consider replacing the default filenames and file extensions associated with key WebSpeed files with proprietary (that is, less easily recognizable) names and extensions that you define. This action limits unauthorized personnel’s ability to recognize these files when they display on the URL.

The following list identifies key files and suggests some ways to shield the identity of each file:

- Rename the default WebSpeed Messenger filename associated with the messenger type that you are using: `cgiip.exe`, `wsisa.dll`, or `wsnsa.dll`.

- Use a file association technique to shield the identity of the default WebSpeed Messenger and broker filenames when they are run. This activity is only supported if you are using a Microsoft Internet Information Web Server (IIS Web Server) in Windows, and your WebSpeed Messenger type is `cgiip.exe`. This technique allows you to define a file extension that can run an executable. The file extension, which includes the default filenames of the WebSpeed Messenger and broker, obscures the identity of these files as it passes the broker name to the executable that runs them.

  For detailed instructions on how to perform this file association technique, refer to the `cgiip.wsc` file that is shipped with the WebSpeed product.

- If you are using UNIX, consider changing the default script name, `wspd_cgi.sh`, to a less immediately identifiable name to hide the WebSpeed messenger and WebSpeed broker names that the `wspd_cgi.sh` file contains.

Restrict your file upload directory

Do not allow execute permissions on your file upload directory.

**Note:** As previously mentioned earlier in this section, do not include references to the file upload directory in your PROPATH.

WebSpeed configuration considerations

As a general guideline to restrict access to your WebSpeed configuration from attack, protect your WebSpeed broker, Web server, and database components inside your firewall. Your Web server and WebSpeed messenger must be on a machine outside this firewall.

**Note:** For more information about security issues as they pertain to WebSpeed configurations and their integration with firewalls, see the “Maximizing WebSpeed compatibility with your firewall” section on page 13–15.
Authenticating a password using SpeedScript

As part of the basic, essential security measures for your site, you should do the following:

1. Ensure that a user is, in fact, a valid user who is recognized by the system.

2. Grant the authenticated user access to only the system resources to which the user is assigned, or authorized to use.

These measures are typically referred to as authentication and authorization, respectively. There are several commonly accepted methods to implement authentication and authorization, such as authentication based on username and password (for which a SpeedScript-based example is provided in this section), standard strong authentication, and Web browser to Web server authentication.

Validating a password using SpeedScript

This section describes how to validate a password in WebSpeed by executing a simple authentication procedure based on username and password. In the procedure, authorization and password variables are passed using an environment variable as written in SpeedScript. To demonstrate the steps in this authentication process, this example is designed to validate the hard-coded value mypass. The Post method is used to protect the value of your password selection.

Note: The Get method is not recommended in this type of authentication process because it automatically displays the value of your password in your browser’s URL field.

To modify this example to query a table that contains user and password information that must be verified:

1. Create a blank HTML file.

2. Enter the following code:

```html
<html>
<head>
<title>Password Example</title>
</head>
<body>
<form method="post" action="val.html"
Enter Password:<input type="password" name="pass" value=""><br>
<input type="submit" name="action" value="process">
</form>
</body>
</html>
```

3. Save and close the file.
4. Create another blank HTML file and enter the following code:

```html
<html>
<head>
<title>Password Page 2</title>
</head>
<body>
<script language="SpeedScript"> 
define variable pass as character.
If get-value("pass") = "mypass" then do:
{&out} "Accepted".
/*Could also run an .html or .r file from here, using the run statement.*/
end.
else do:
{&out} "Denied".
/*Could also redirect the user to an .html or .r file from here, using the run statement.*/
end.
</script>
</body>
</html>
```

5. Close and save this second file with the filename `val.html`.

6. Run the first HTML file you created in **Step 2**.

   The word **Accepted** appears to verify the acceptance and validation of the username and password values.
Securing data transmissions between WebSpeed client and server components

Transmitting data over the Internet exposes your WebSpeed site to a host of potentially hostile activities. By using the Secure HTTP Protocol (HTTPS) and the Secure Sockets Layer (SSL), you can transmit data to and from external destinations in a secure manner. The sections that follow briefly discuss how your WebSpeed applications can benefit from each of these industry-accepted protocols.

If you intend to use secure transmissions, it is important that you read OpenEdge Getting Started: Core Business Services for a more detailed discussion of these technologies. This manual explains the relevant concepts and gives instructions for using the keys, certificates, and management tools that OpenEdge provides.

Secure HTTP Protocol (HTTPS)

The Secure Hypertext Transport Protocol, also referred to as HTTPS, is a security-enhanced extension of the Hypertext Transmission Protocol (HTTP). The HTTPS is designed to support transmission of individual messages, or Web pages, securely over the Internet.

HTTPS adds application-level encryption and security on top of ordinary sockets-based communications. The client and server communicate over any ordinary HTTP session and then negotiate their security requirements. They can use public and private keys to encrypt and de-encrypt messages transmitted via HTTPS.

HTTPS provides many benefits for WebSpeed users to support their e-commerce efforts; encryption, authentication, and message integrity are just a few of the benefits of using this protocol.

WebSpeed Messengers are compatible with HTTPS; however, configuration of your system to use HTTPS between browser clients and the Web server does not involve any OpenEdge components. Refer to your Web server documentation to configure the Web server for HTTPS communications.

Secure Sockets Layer (SSL)

SSL has been widely accepted on the Internet as the preferred, secure protocol to authenticate and encrypt communication between clients and servers. Where HTTPS only secures individual messages, SSL secures the points of origin and destination for any amount of data that can be transmitted between a client and server. This protocol further complements and extends the security measures available to protect the confidentiality of your communications.

The SSL protocol resides above the network protocol, as defined by Transmission Control Protocol/Internet Protocol (TCP/IP), and directly below the application protocols, such as HTTP, HTTPS, or IMAP. It uses TCP/IP on behalf of the application protocols and, in the process, enables the following activities to occur:

- An SSL-enabled server can authenticate itself to an SSL-enabled client
- An SSL-enabled client can authenticate itself to an SSL-enabled server
- The SSL-enabled server and client can establish an encrypted connection

OpenEdge supports SSL communications between the WebSpeed Messenger and the WebSpeed Transaction Server, as described in the next section.
SSL-enabled WebSpeed Transaction Server operation

You have the option of configuring any WebSpeed Transaction Server instance to require Secure Sockets Layer (SSL) client connections. You can maintain both SSL-enabled and non-SSL Transaction Server instances, but a given instance supports only one type of connection, either secure or non-secure.

Security derives from the client authentication of the server’s identity via a Public Key Infrastructure (PKI) and a symmetric data encryption system. To configure a Transaction Server instance for SSL operation, you must:

- Obtain and install a server private key and a public key certificate. OpenEdge provides built-in keys and certificates that are suitable for use on development or demonstration servers; for production machines, you should obtain server certificates from an internal or public Certificate Authority (CA).
- Specify an alias and password for access to the private key/digital certificate.
- Disable session caching, or enable it with a specified timeout.

To connect to an SSL-enabled WebSpeed Transaction Server, a WebSpeed Messenger must have access to a digital (public key) certificate (often called a CA Root Certificate) that can authenticate with the digital certificate used by the server, and the Messenger must be configured to send SSL requests.

To perform these configuration tasks, you can use OpenEdge Explorer or manually edit the ubroker.properties file, as explained in the next section.

SSL configuration for WebSpeed components

To enable SSL communications, you must configure both the WebSpeed Messenger and the WebSpeed Transaction Server. Configure the Messenger as follows:

- The Messenger must be SSL-enabled, meaning that it sends SSL data to the Transaction Server that is to process the client requests. To configure the Messenger to send SSL requests, you set the property sslEnable=1. You set this property by checking the Enable SSL AppServer connections box in the SSL properties category in OpenEdge Explorer, or by manually editing the ubroker.properties file. In addition, you must obtain and install public key certificates for the Messenger host machine.
- Determine whether you want the Messenger to verify the host name for the WebSpeed Transaction Server by comparing it with the Common Name specified in the server digital certificate, and raise an error if they do not match (the default behavior). You can disable this verification by setting the property noHostVerify=1. To do so, check the Disable Client Verification of SSL Host Name box in OpenEdge Explorer, or manually edit the ubroker.properties file. In addition, you must obtain and install public key certificates for the Messenger host machine.
- Determine whether you want the Messenger to request reuse of the session ID for successive connections to the same Transaction Server (the default behavior). If not, set the property noSessionReuse=1, either by checking the Disable SSL Session Reuse box in OpenEdge Explorer or by editing the ubroker.properties file. (The default behavior does not guarantee that session IDs can be reused, because the server might disallow session reuse.)
Configure the Transaction Server as follows:

- The Transaction Server must be SSL-enabled, meaning that it accepts SSL requests from the Messenger. You set the property `sslEnable=1` by checking the Enable SSL Client Connections box in the SSL General properties category in OpenEdge Explorer, or by manually editing the `ubroker.properties` file. You must also obtain and install a server private key and public key certificate, unless you are using the defaults provided with OpenEdge.

- In the SSL General properties category in OpenEdge Explorer, select the alias for the private key/digital certificate entry (in the OpenEdge keystore) that you want to secure connections for this Transaction Server. Also enter and confirm the password for this private key and digital certificate. You need not enter a password if you choose to use the `default_server` certificate and its default password. (Note: The password is encrypted in the `ubroker.properties` file; if manually editing the file, you must use the `genpassword` utility to encrypt the password. The properties appear in the file as `keyAlias=` and `keyAliasPasswd=`.)

- In the SSL Advanced Features properties category in OpenEdge Explorer, you can enter a timeout value that specifies the length of time (in seconds) that a disconnected session is held in the cache. During this specified interval, a connected client can resume its session. To disable session caching, check the box, or edit the `ubroker.properties` file and set the property `noSessionCache=0`. The timeout value appears in the file as `sessionTimeout=n`.

For more information on setting properties for WebSpeed Messengers and Transaction Servers and other Unified Broker products, see the OpenEdge Explorer help or the `OpenEdge-Install-Directory\properties\ubroker.properties.README` file.
Maximizing WebSpeed compatibility with your firewall

This section reviews ways to enhance your site interaction with your firewall.

The main purpose of a firewall is to restrict access to your Web site to ensure that only authorized, authentic users are allowed to connect through specific communication access, or controlled, points. The following topics highlight important security issues related to OpenEdge and WebSpeed communication ports and traffic that interact with the firewall:

- WebSpeed TCP port details
- Important firewall considerations
- Network Address Translation (NAT) and the WebSpeed distributed configuration
- Using the NameServer client port range value settings with a firewall

WebSpeed TCP port details

Your WebSpeed configuration might have specific firewall configuration requirements, especially when the WebSpeed Messenger is on a separate machine from the WebSpeed broker. If a firewall exists between any two WebSpeed components, the firewall must be opened on the appropriate port.

Table 13–3 identifies the network communications requirements for your WebSpeed configuration.

Table 13–3: Network communications requirements

<table>
<thead>
<tr>
<th>Communication</th>
<th>Basic requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web browser to Web server</td>
<td>The Web browser must have access to the Web server. Typically, port 80 is used for this purpose; however, you can set up any port.</td>
</tr>
<tr>
<td>WebSpeed Messenger to NameServer</td>
<td>The WebSpeed Messenger communicates with the NameServer using the User Datagram Protocol (UDP) that is open inbound on port 5162, or the NameServer port number. UDP ports also must be open outbound from the NameServer to the WebSpeed Messenger to complete the communication process.</td>
</tr>
<tr>
<td>WebSpeed Messenger to WebSpeed broker</td>
<td>The WebSpeed Messenger communicates with the WebSpeed broker on a specific port that is defined when the WebSpeed broker is configured. You can check this information either in OpenEdge Explorer or in the <code>install-dir/properties/ubroker.properties</code> file.</td>
</tr>
<tr>
<td>WebSpeed Messenger to WebSpeed agents</td>
<td>The WebSpeed Messenger also communicates directly with WebSpeed agents. When you configure the WebSpeed broker, you must specify a port range for the agents to use. You can check this information either in OpenEdge Explorer or in the <code>install-dir/properties/ubroker.properties</code> file.</td>
</tr>
<tr>
<td>WebSpeed agents to database</td>
<td>WebSpeed agents are ABL clients. Use the OpenEdge client-server configuration to establish network communication between these clients and their associated database.</td>
</tr>
</tbody>
</table>
Important firewall considerations

The following information highlights key considerations regarding WebSpeed and firewall configuration:

- It is not advisable to set up a firewall between WebSpeed and a remote database. This type of arrangement can unnecessarily expose your site to security leaks on many levels. It is preferable that you set up your WebSpeed brokers and agents with your database behind the same firewall.

- Avoid opening the database directly to the Internet.

Network Address Translation (NAT) and the WebSpeed distributed configuration

Network Address Translation, or NAT, is an Internet standard that supports one set of Internet Protocol (IP) addresses for external traffic and a second set of addresses for internal use. A router translates these IP addresses between network domains, hiding internal addresses and generating globally unique, routable addresses. NAT, and IP Convert, are general forms to mask IP addresses, shielding this information from the Internet. In this manner, the NAT process helps promote more secure connections between clients and WebSpeed.

OpenEdge and WebSpeed support NAT in a distributed configuration arrangement through a registration process. The value you select for the Registration Mode indicates how the WebSpeed broker identifies its hostname when the WebSpeed broker registers with its controlling NameServer. When a client application attempts to connect to an application service on a host machine that the WebSpeed broker supports, the hostname information is passed to the client application, provided that the information can be authenticated.

The specific value you set up for the Registration mode depends on the connection arrangement you want. The following procedures identify and describe the Registration mode values.
To make a client connection through a NAT firewall using OpenEdge Explorer:

1. Double click the WebSpeed folder on the left-hand side of the main OpenEdge Explorer window.

2. Select the broker name.

3. Choose Configuration and then Edit.

4. Select the Broker tab to see the Controlling NameServer Information.

5. Select the Registration Mode value you want to set. Table 13–4 defines each value.

6. Click OK.

<table>
<thead>
<tr>
<th>Select this Registration Mode . . .</th>
<th>To Identify the hostname as . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register-IP (default)</td>
<td>The IP address of the machine where the WebSpeed broker is located. The WebSpeed broker determines its IP address and passes this information to the NameServer when the broker registers. Clients connect to the host using the broker’s IP address.</td>
</tr>
<tr>
<td>Register-LocalHost</td>
<td>The hostname of the machine where the WebSpeed broker is located. The WebSpeed broker determines its hostname and passes this information to the NameServer when the broker registers. Clients connect to the host using this hostname.</td>
</tr>
<tr>
<td>Register-Host Name (as defined in the unlabeled property description field)</td>
<td>The IP address defined. The WebSpeed broker passes the IP address value defined in the property description field to the NameServer when the broker registers. Clients connect to the host using the IP address defined.</td>
</tr>
</tbody>
</table>
To make a client connection through a NAT firewall using the ubroker.properties file:

1. Search the file for the broker property section, or sections, that you want to change.

2. Set the value of the registrationMode property. Table 13–5 defines the values from which you can choose to enter in the registrationMode property.

### Table 13–5: RegistrationMode Connection—ubroker.properties file

<table>
<thead>
<tr>
<th>Enter this Registration Mode . . .</th>
<th>To Identify . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Register-IP</strong> (default)</td>
<td>The IP address of the machine where the WebSpeed broker is located. The WebSpeed broker determines its IP address and passes this information to the NameServer when the broker registers. Clients connect to the host using the broker’s IP address.</td>
</tr>
<tr>
<td><strong>Register-LocalHost</strong></td>
<td>The hostname of the machine where the WebSpeed broker is located. The WebSpeed broker determines its hostname and passes this information to the NameServer when the broker registers. Clients connect to the host using this hostname.</td>
</tr>
</tbody>
</table>
| **Register-HostName**              | A specific a hostname or IP address. You must:

  - Enter the value Register-HostName in the registrationMode property field.
  - Enter the hostname, or the IP address of the hostname, in the hostName property field to identify where the WebSpeed broker is running.
  - Enter a description of the type of IP address you entered in the description property field.

  The WebSpeed broker passes either the hostname or the IP address value defined in the hostName property to the NameServer when the broker registers. Clients connect to the host using either the hostName or the IP address defined. |

**Note:** For more information, see the “GENERAL INSTRUCTIONS for configuring the Unified Broker and NameServer for WebSpeed and AppServers” in the ubroker.properties file.
Using the NameServer client port range value settings with a firewall

The WebSpeed Messenger communicates with the NameServer using the User Datagram Protocol (UDP). When the WebSpeed Messenger initiates the connection, it sends a UDP packet to the known NameServer port. When the NameServer receives the packet, it sends a response packet, also in UDP packet form, back to the WebSpeed Messenger.

To facilitate communication when a firewall exists between the WebSpeed Messenger and the NameServer, you could open all UDP ports from the machine that is running the NameServer to the machine that is running the WebSpeed Messenger.

However, opening all 65,000 UDP ports from inside the firewall to outbound ports is a time-consuming job. Also, opening every port is not a necessity. You can specify a client port range minimum with minNSClientPort and a client port range maximum with maxNSClientPort.

Using these two parameters, a firewall administrator can restrict the UDP response from the NameServer to the client. The administrator can specify a range of ports in the properties file and therefore reduce the number of UDP ports that are open in the firewall.

The following rules apply to the values set for these parameters:

- The value for these two parameters must be a number between 1024 and 65535 inclusive (or 0).
- The minNSClientPort also must be less than maxNSClientPort.
- If both minimum and maximum values are set to zero, then a random port number will be dynamically assigned. This is the default setting. The assigned port number will be in the range of 1024 through 65535.
- If both minimum and maximum values are set to the same number, the port number will be used exclusively for NameServer communication.
This chapter contains information for developers who want to use Microsoft’s Active Server Page (ASP) technology in conjunction with their WebSpeed applications, as described in the following sections:

- Setting up the WebSpeed ASP (WSASP) Messenger
- Accessing the WebSpeed ASP Web page
- Enabling the WebSpeed ASP example
Setting up the WebSpeed ASP (WSASP) Messenger

The WebSpeed Active Server Page (WSASP) Messenger allows you to integrate WebSpeed applications with Microsoft Active Server Pages. This integration allows you to use Microsoft’s server side scripting and ActiveX server components.

In order to enable the use of Active Server Pages, you must set up the WSASP Messenger on a machine that has WebSpeed installed and is running a Microsoft Web server.

To set up the WSASP Messenger:

1. Log in as administrator on the machine that is running the Web server.
2. Shut down the Web server.
3. Register the `wsasp.dll` file:
   a. Open a DOS window.
   b. Change directory to `install-path/bin`.
   c. Enter: `regsvr32.exe wsasp.dll`.
4. Restart the Web server.
5. Test your configuration.

You can test your configuration (and also customize error messages) from the WebSpeed Configuration and Verification Page. A link to the WebSpeed Configuration and Verification Page is on the WebSpeed ASP Web Page.
Accessing the WebSpeed ASP Web page

The WebSpeed ASP Web page allows you to:

- Test the WSASP Messenger
- Customize error messages
- View documentation on the properties and methods of the WSASP messenger
- Run a sample application

To access the WebSpeed ASP Web page, you must create a virtual directory called `wsasp` from the Microsoft Web server console. The virtual directory must point to `install-path/webspeed/wsasp` and must have execute permissions.

When you go to `http://host-name/wsasp`, you should see the page shown in Figure 14–1.
Enabling the WebSpeed ASP example

Before you can run the ASP example, you must complete some WebSpeed configuration tasks.

To configure WebSpeed to run the ASP example:

3. Start a WebSpeed broker named wsbroker1 whose agents are connected to the Sports2000 database server.
4. Make wsbroker1 the default service.

You can do this in OpenEdge Explorer by selecting Application service names on wsbroker1. Make sure that the Supports default service check box is checked.
Part V

Messaging and ESB Administration

Chapter 15, OpenEdge Adapter for SonicMQ Administration

Chapter 16, Configuring and Managing the OpenEdge Adapter for Sonic ESB
OpenEdge Adapter for SonicMQ Administration

This chapter provides instructions for administration and configuration tasks associated with the OpenEdge Adapter for SonicMQ®, as well as instructions for working with the sample applications installed with OpenEdge, as described in the following sections:

- Introducing the OpenEdge Adapter for SonicMQ
- Configuring and administering the OpenEdge Adapter for SonicMQ
- Maximizing performance
- Setting the CLASSPATH
- Internationalization considerations
Introducing the OpenEdge Adapter for SonicMQ

The OpenEdge Adapter for SonicMQ enables ABL client applications to function within a SonicMQ enterprise messaging system. It provides an ABL-to-JMS API that translates native ABL code to Java Message Service (JMS) protocol and vice versa. JMS is the standard communication protocol for the exchange of messages among applications in the SonicMQ environment.

For more information on OpenEdge Adapter for SonicMQ architecture, see OpenEdge Getting Started: Application and Integration Services.

Client connections to the OpenEdge Adapter for SonicMQ

The OpenEdge Adapter for SonicMQ allows OpenEdge applications to communicate via JMS Messaging through SonicMQ.

**OpenEdge Adapter for SonicMQ BrokerConnect (BrokerConnect)**

The ABL client application connects to BrokerConnect by specifying, when the JMS session is created, the connection parameters of either the NameServer (if a controlling NameServer is specified for the adapter) or the adapter itself.

The client can connect in the following ways:

- Via HTTP or HTTPS over the Internet, with use of the AppServer Internet Adapter. See Chapter 3, “Configuring and Managing the AppServer Internet Adapter,” for more information.

- Via AppServer protocol, with or without Secure Sockets Layer (SSL) tunneling. To support SSL communications with client applications, BrokerConnect must be configured as an SSL-enabled server. See the “SSL-enabled BrokerConnect operation” section on page 15–3 for more information.

**OpenEdge Adapter for SonicMQ ClientConnect (ClientConnect)**

ClientConnect is for OpenEdge clients and will run as a background process in conjunction with an ABL client. There is a single adapter process per client process with the SonicMQ Broker acting as a service point for all JMS sessions.

**OpenEdge Adapter for SonicMQ ServerConnect (ServerConnect)**

The ServerConnect option is for OpenEdge Application Servers (WebSpeed and AppServer). With this configuration, there is a single adapter process per unified broker process, allowing multiple Application Server agents to connect to this single adapter process.

For information about programming ABL client applications in the SonicMQ environment, see OpenEdge Development: Messaging and ESB. Also see OpenEdge Getting Started: Application and Integration Services for a discussion of OpenEdge Adapter for SonicMQ architecture.
Installing the OpenEdge Adapter for SonicMQ

For information on installing the OpenEdge Adapter for SonicMQ, see *OpenEdge Getting Started: Installation and Configuration*.

SSL-enabled BrokerConnect operation

You have the option of configuring BrokerConnect instance to require SSL client connections. You can maintain both SSL-enabled and non-SSL BrokerConnect instances, but a given instance supports only one type of connection, either secure or nonsecure.

Security derives from the client authentication of the server’s identity via a Public Key Infrastructure (PKI) and a symmetric data encryption system. To configure an adapter instance for SSL operation, you must:

- Obtain and install a server private key and a public key certificate. OpenEdge provides built-in keys and certificates that are suitable for use on development or demonstration servers; for production machines, you should obtain server certificates from an internal or public Certificate Authority (CA).

- Specify an alias and password for access to the private key/digital certificate.

- Disable session caching, or enable it with a specified time-out.

To perform these configuration tasks, you can use OpenEdge Explorer or manually edit the `ubroker.properties` file. You can use the `mergeprop` utility installed with OpenEdge to manually edit the `ubroker.properties` file. For information on using `mergeprop`, see *OpenEdge Getting Started: Installation and Configuration*.

To connect to an SSL-enabled BrokerConnect, a client application must have access to a digital (public key) certificate (often called a CA Root Certificate) that can authenticate with the digital certificate used by the adapter, and the client must use a secure protocol.

For more information on SSL support in OpenEdge, see *OpenEdge Getting Started: Core Business Services*.

HTTPS security for BrokerConnect

BrokerConnect allows ABL applications to communicate with other applications using the Java Message Service (JMS) infrastructure managed by SonicMQ. All security for the JMS infrastructure accessed by the OpenEdge Adapter for SonicMQ is provided by SonicMQ. For more information, see the SonicMQ documentation.

OpenEdge supports data privacy using SSL for the connection between the ABL client and BrokerConnect directly over intranet connections and over the Internet through the AppServer Internet Adapter (AIA) using HTTPS. You can also secure the direct connection between an AIA and BrokerConnect server session using SSL. BrokerConnect allows you to set SSL server session properties for BrokerConnect using the Progress Unified Broker framework. This is similar to setting SSL server session properties for an AppServer. For more information on setting SSL for BrokerConnect, see the “Configuring BrokerConnect” section on page 15–5. For more information on SSL support in OpenEdge, see *OpenEdge Getting Started: Core Business Services*. For more information on configuring the AIA for Internet access to OpenEdge server sessions, see the “Configuring an AIA with OpenEdge Explorer” section on page 3–8.
Configuring and administering the OpenEdge Adapter for SonicMQ

The following sections describe:

- Configuring ClientConnect
- Configuring ServerConnect
- Configuring BrokerConnect

Configuring ClientConnect

ClientConnect is a process that runs with your ABL client session. In the following example, the application creates a session procedure by calling `jmssession.p` persistently specifying the `-SMQConnect` connection parameter:

Using SMQConnect on a client example

```
DEFINE VARIABLE sessionH AS HANDLE.
RUN jms/jmssession.p PERSISTENT SET sessionH ("-SMQConnect").
RUN setBrokerURL IN sessionH ("MQBrokerHost:2506").
RUN beginSession IN sessionH.
```

The only configuration is for logging and debugging, if necessary. By default, logging is disabled for ClientConnect sessions. Logging is turned on by setting the `brkrLoggingLevel` and `srvrLoggingLevel` properties to a value greater than 0, and setting `brkrLogEntryTypes=UBroker.Basic` (the default).

**Note:** Logging level represents the amount of logging information written to the log file. Valid logging level values are 1 (Errors), 2 (Basic), 3 (Verbose), and 4 (Extended). The default value is 2. For more information on logging, see *OpenEdge Development: Debugging and Troubleshooting*.

The property and logging options for ClientConnect are stored in the `Adapter.CC.cc1` section of the `install-dir/properties/JavaTools.properties` file. These properties must be modified manually, and are applicable to all OpenEdge clients using ClientConnect functionality.

Configuring ServerConnect

ServerConnect is a process that runs with your AppServer ABL session or WebSpeed SpeedScript session. The ServerConnect process that runs inside of the AppServer or WebSpeed server is multi-threaded and allows for multiple SonicMQ connections within the same process. Additionally, each connection to a SonicMQ Broker uses multiple threads.

Prior to using ServerConnect in an AppServer or WebSpeed server, the AppServer or WebSpeed server must be enabled for **SonicMQ ServerConnect enabled** using OpenEdge Explorer.
To enable the AppServer or WebSpeed server for ServerConnect:

1. Select the Messaging tab for the AppServer or WebSpeed server.
2. Select the SonicMQ ServerConnect enabled check box, as shown:
3. Select unique broker and server log filenames.
4. Select the logging level.

These settings start a SonicMQ ServerConnect process when the AppServer or WebSpeed server starts with specified logging options. After starting the AppServer or WebSpeed server, ensure the SonicMQ Broker is running.

The only necessary configuration is for logging and tuning. Logging properties are defined through the AppServer or WebSpeed broker. These settings can be modified using OpenEdge Explorer.

To configure ServerConnect logging for an AppServer, see Chapter 2, “Configuring and Managing the AppServer.” To configure ServerConnect logging for a WebSpeed Transaction Server, see Chapter 9, “Configuring WebSpeed in Windows.”

Configuring BrokerConnect

BrokerConnect is a Unified Broker product and part of the Progress Unified Broker framework. In Windows, you can use OpenEdge Explorer to start, stop, get status, add, delete, and edit properties of BrokerConnect.

To define and configure a BrokerConnect instance:

1. Make sure that the AdminServer is running on the host where you want to configure the BrokerConnect instance.
2. Open OpenEdge Explorer from the OpenEdge program group.
3. Connect to the AdminServer on your BrokerConnect host.
4. To define a new adapter instance or to modify an existing BrokerConnect configuration, do one of the following:
   - To define a new adapter instance, select the SonicMQ Adapter folder in the OpenEdge Explorer’s tree view and click Configuration and Edit. Enter a unique name for the adapter instance. Choose Save. Set any global preferences that you require (see the online help), and choose Save.
   - To modify an existing BrokerConnect configuration, expand the SonicMQ Adapter folder in the tree view, select the adapter instance you want to modify, and click Configuration and Edit.
5. Select a property category tab and set the properties as required. You can accept the default values, if they are appropriate for your application.

For information on setting the CLASSPATH, see the “Setting the CLASSPATH” section on page 15–12.
Expanding the Broker category shows the following property subcategories:

- **General** — Specify a working directory and the TCP/IP port number where BrokerConnect listens for requests. Check the Auto start box if you want the adapter to start whenever you start the AdminServer.

- **Owner Information** — You can optionally provide Username and Password information for the user who owns the BrokerConnect instance.

- **Controlling NameServer** — If you plan to use a NameServer to control BrokerConnect access, check the Register with NameServer box and select a controlling NameServer from the drop-down list. Select one of the three Registration Mode options to specify how you want the NameServer to provide the host address of BrokerConnect for client connections.

**Note**: If BrokerConnect is not registered with a controlling NameServer, clients connect directly to the adapter by specifying its TCP/IP host and port with the -DirectConnect parameter.

- **AppService Name List** — Enter the names of the application services that the BrokerConnect instance is to register with the controlling NameServer, if any. Clients connecting to the adapter must specify one of these service names.

- **Logging Setting** — Specify a pathname for the broker log file, the level of logging detail, and whether the logging for a session appends to or overwrites the previous broker log file.

- **Advanced Features** — Specify the maximum number of client connections (Maximum client instances) that the BrokerConnect can support at one time. Also specify an interval in seconds (Registration retry) to set the frequency with which the adapter sends a “keep-alive” message to tell the NameServer that it is active.

6. Select the Server tab to display the following properties:

- **General** — Specify any startup parameters for BrokerConnect.

- **Logging Setting** — Specify a pathname for the server log file, the level of logging detail, and whether the logging for a session appends to or overwrites the previous broker log file.

- **Pool Range** — These settings determine the number of threads that the BrokerConnect agent can start up and maintain. One thread is required for each active client application.

7. Select the SSL category to show the following property subcategories:

- **General** — If you check the Enable SSL client connections box, select the alias for the private key/digital certificate entry (in the OpenEdge keystore) that you want to secure connections for this adapter instance. Also enter and confirm the password for this private key and digital certificate. You need not enter a password if you choose to use the default_server certificate and its default password.
• **Advanced Features** — By default, caching is enabled for the SSL client session, and you can enter a time-out value that specifies the length of time (in seconds) that a disconnected session is held in the cache. During this specified interval, a connected client can resume its session. To disable session caching, check the box.

The options in the SSL category options define the security settings for an SSL-enabled BrokerConnect instance. Note that a BrokerConnect enabled for SSL operation does not accept non-SSL client connections. For more information on SSL operations, see *OpenEdge Getting Started: Core Business Services*.

8. Select the **Environment Variables** if you want to specify environment variables for BrokerConnect execution. It allows you to enter name-value pairs for environment variable settings. Any values you set here override prior values set for the same environment variables in the operating system.

9. Click **OK** to save the configuration.

**Editing BrokerConnect attributes in ubroker.properties**

The **Adapter** root group in the *ubroker.properties* file supports the instance of the SonicMQ Broker for BrokerConnect. You can configure BrokerConnect by manually editing this file.

**Guidelines for editing the properties file**

When editing the *ubroker.properties* file without OpenEdge Explorer, note the following:

• You should **not** directly change the values in the *ubroker.properties* file unless you have a complete understanding of how the changes affect components. When possible, always use OpenEdge Explorer to make all changes to this file.

**Note:** You can use the `mergeprop` utility installed with OpenEdge to manually edit the *ubroker.properties* file. For information on using `mergeprop`, see *OpenEdge Getting Started: Installation and Configuration*.

• Always make a copy of this file, edit the copy, and verify the result before replacing the original with your edited copy.

• For complete definitions of all the properties and detailed information on how to set them, see the *ubroker.properties*.README file, as well as the comments included in the properties file itself. Both files reside in the *properties* directory.

The 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 products or product instances.
AppServer entries in the properties file

BrokerConnect configurations in ubroker.properties can include the entities listed in Table 15–1.

### Table 15–1: AppServer configuration entity names

<table>
<thead>
<tr>
<th>Configuration entity name</th>
<th>Configuration entity function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Adapter]</td>
<td>Defines default property settings for all BrokerConnect instances.</td>
</tr>
<tr>
<td>[Adapter.product-instance-name]</td>
<td>Defines property settings for the BrokerConnect instance. The ubroker.properties file can contain several of these entities, each with a unique <code>product-instance-name</code>. For BrokerConnect, <code>product-instance-name</code> is the name you specify for the adapter instance.</td>
</tr>
</tbody>
</table>

Server startup parameters

You can optionally specify the attributes described in Table 15–2 in the `srvrStartupParam` property of BrokerConnect.

### Table 15–2: BrokerConnect attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>brokerURL</td>
<td>String</td>
<td>Null</td>
</tr>
<tr>
<td>user</td>
<td>String</td>
<td>Null</td>
</tr>
<tr>
<td>password</td>
<td>String</td>
<td>Null</td>
</tr>
<tr>
<td>clientID</td>
<td>String</td>
<td>Null</td>
</tr>
<tr>
<td>pingInterval</td>
<td>Number (of seconds) controlling how frequently the adapter instance pings the broker to verify that the broker is responding</td>
<td>None—the adapter instance does not ping the broker unless an interval is specified</td>
</tr>
</tbody>
</table>

**Note:** The `srvrStartupParam` property is a dynamic property. Any changes to this property only affects new agents started after the value changes because the change affects how the agent is started. Existing (running) brokers and agents are not updated with the new value.

If `srvrStartupParam` attributes are specified, they serve as default values for all of the clients; however, the ABL-JMS API allows clients to overwrite the `srvrStartupParam` defaults.

Names of attributes are case-sensitive and must be separated with a semicolon (;). For example:

```
srvrStartupParam=brokerURL=localhost; user=u1; password=p1;
```
Configuring BrokerConnect from the command line

You can use two command-line tools, adaptconfig and adaptman, with BrokerConnect on all OpenEdge-supported platforms.

Adaptconfig

Use adaptconfig to validate manual changes you made to the ubroker.properties file for BrokerConnect instances. This is the adaptconfig syntax:

**Syntax**

```
```

For information on using the adaptconfig utility, see the “ADAPTCONFIG” section on page B–3.

Adaptman

Use adaptman to start, stop, query, and kill an existing instance of a SonicMQ Broker for BrokerConnect or to manipulate brokers on other machines by specifying the name of the machine and the port the AdminServer is running on. This is the adaptman syntax:

**Syntax**

```
adaptman { { -name adapter-broker { -kill | -start | -stop | -query } [ -host host-name -user user-name | -user user-name ] [ -port port-number ] } | -help }
```

For information on using the adaptconfig utility, see the “ADAPTMAN” section on page B–4.

For information on serialized connections and administered objects, see *OpenEdge Development: Messaging and ESB*. 
Maximizing performance

The primary goal of a JMS messaging system is to reliably distribute asynchronous business events and information between applications. This is achieved by a loosely coupled communication style of application integration. A more tightly coupled communication mechanism, such as sockets or direct calls to the AppServer, is useful for passing large amounts of data or for subsecond response time.

Performance comparison

The following example describes the kind of performance you can expect. It compares passing data between two ABL clients through a JMS server with passing the same data between two ABL clients through an AppServer application.

This configuration includes:

- Two ABL clients on a Solaris SPARC 20 machine
- SonicMQ broker on a Windows NT 300MHz machine on the LAN

The first client publishes the customer table of the Sports database as a StreamMessage with each record written as a bytes item using RAW-TRANSFER. The second client subscribes to the JMS server, receives the message, and puts the data in a temp-table. It takes, on average, 1.5 seconds to transfer the table.

Passing the customer table from one client to another through the AppServer (by passing it from the first client as an input temp-table to an AppServer application and then passing it to the second client, from the AppServer application, as an output temp-table) takes, on average, 1.3 seconds.

Optimizing message size

When performance is an issue, fewer and larger messages perform better than many small messages. The optimal message size is several thousand bytes.

StreamMessage, MapMessage, and TextMessage

StreamMessages and MapMessages consist of individual items (or chunks) of data. The larger the items are, the better the performance is. For example, a group of database records can be sent in a StreamMessage with each field as a separate item (using a write... method). Much better performance is achieved if each record is converted to RAW data and written as a Byte item in a StreamMessage using writeBytesFromRaw(). Applications can use multiple appendText methods to generate larger messages in a TextMessage and XMLMessage. Using larger segments in each appendText improves performance.

Remote and local calls

In general, local ABL-JMS API calls are less expensive than remote calls (calls that go to the OpenEdge Adapter for SonicMQ and the SonicMQ broker). Those procedures and functions that execute remotely are noted in OpenEdge Development: Messaging and ESB. This information is useful when analyzing the performance of an application.
Message reuse

The creation of an ABL message is relatively expensive. The publisher (or sender) of a message should reuse a Message object whenever possible. The message can be cleared for reuse by calling clearBody and clearProperties. The message body of some message types is automatically cleared when new data is set. (For more information, see OpenEdge Development: Messaging and ESB.

The application that consumes messages can reuse them by calling the setReuseMessage message consumer method. If setReuseMessage is called, the message consumer reuses the same Message object for all the messages it receives, provided that the message was not deleted by the application.

Load balancing

SonicMQ supports client-side load balancing. With this enabled, a connect request can be redirected to another broker within the SonicMQ cluster, provided broker-side load balancing has not been disabled.

Client-side load balancing involves the following methods on the session handle:

- setLoadBalancing
- getLoadBalancing

For more information on these methods, see OpenEdge Development: Messaging and ESB.

Discardable messages

When you publish a message to a topic, you can specify the DISCARDABLE delivery mode. If you do and the destination message queue is full, the message is automatically discarded.

You can specify the DISCARDABLE delivery mode in the following methods:

- setDefaultPersistency
- publish

For more information on discardable messages, see the reference entries for these methods in OpenEdge Development: Messaging and ESB.

For more information on maximizing performance, refer to SonicMQ Performance Tuning Guide.
Setting the CLASSPATH

In Windows and on UNIX, you can set the CLASSPATH by using the PluginPolicy.Progress.SonicMQ section in the AdminServerPlugins.properties file. BrokerConnect uses the pluginclasspath property. ClientConnect and ServerConnect use the classpath property.
Internationalization considerations

The ABL interpreter (for the client, AppServer, and WebSpeed) supports many code-page encoding standards. The JMS client uses Unicode. The translation of text data between the ABL’s code page and Unicode is done automatically by the ABL-JMS implementation (for more information, see *OpenEdge Development: Messaging and ESB*).

When an ABL client sends text data to JMS (for example, in a TextMessage or a StreamMessage), the ABL client must send the text in a Unicode/UTF-8 format. If the internal code page of the client is not in Unicode/UTF-8 format (-cpinternal UTF-8), the ABL-JMS implementation must convert the text to UTF-8.

When text is converted to UTF-8, each character can require up to three bytes. This causes the text size limit of each text chunk to be 10K, since the conversion routine must prepare enough expansion room. Since all the message types support segmentation of text data, the limit can be worked around by using multiple segments. Whenever possible, the ABL client’s internal code page should be set to UTF-8 to avoid performing code-page conversions and to eliminate the 10K size limit.
Configuring and Managing the OpenEdge Adapter for Sonic ESB

This chapter contains instructions for managing OpenEdge services in the Sonic ESB environment, as described in the following sections:

- Introduction to the OpenEdge Adapter for Sonic ESB
- Installation of the OpenEdge Adapter for Sonic ESB
- Using the OpenEdge Adapter for Sonic ESB
- Exposing a service as a standard Web service
- Deploying a service instance in Sonic ESB
- Security considerations for OpenEdge Adapter for Sonic ESB
Introduction to the OpenEdge Adapter for Sonic ESB

The OpenEdge Adapter for Sonic ESB® makes it possible to deploy AppServer applications in Sonic ESB. The OpenEdge Adapter for Sonic ESB provides two methodologies for exposing an ABL application as a service in Sonic ESB:

- **Native Invocation** — Using the Native Invocation methodology, ABL procedures are called directly from an ESB process.

- **Web Service Invocation** — Using the Web Service Invocation methodology, the OpenEdge Adapter for Sonic ESB converts SOAP messages to AppServer protocol on inbound client requests, and conversely converts AppServer protocol to SOAP on messages returned from the service.

As a Sonic ESB service, an AppServer application can be:

- Fully integrated in ESB processes

- Called natively with the Native Invocation methodology, or exposed as a standard Web service with the Web Service methodology

- Managed with the use of Sonic’s powerful management and customization tools

This chapter focuses on management of the OpenEdge Adapter for Sonic ESB and the services that it supports. For a description of basic architecture and how the OpenEdge Adapter for Sonic ESB operates, see *OpenEdge Getting Started: Application and Integration Services*. For additional information, including guidelines for developing ABL applications for use as Sonic ESB services, see *OpenEdge Development: Messaging and ESB.*
Installation of the OpenEdge Adapter for Sonic ESB

Installation of the OpenEdge Adapter for Sonic ESB is fully automated in those cases where the Sonic Domain Manager is installed and running on the same machine where OpenEdge is installed. There are three different types of installations possible, and the type of installation is determined by a combination of the OpenEdge products selected to be installed and the Sonic products currently installed on the machine:

- **Domain** — Installed when the Sonic Domain Manager is installed and the user has selected only the OpenEdge Adapter for Sonic ESB product

- **Deployment** — Installed when Sonic Workbench is **not** detected on the installation machine and the user has selected one of the following products to install in addition to the OpenEdge Adapter for Sonic ESB:
  - OpenEdge Application Server Basic
  - OpenEdge Application Server Enterprise
  - OpenEdge Studio
  - OpenEdge Architect
  - 4GL Development

- **Development** — Installed when Sonic Workbench is installed and the user has selected one of the following products to install in addition to the OpenEdge Adapter for Sonic ESB:
  - OpenEdge Application Server Basic
  - OpenEdge Application Server Enterprise
  - OpenEdge Studio
  - OpenEdge Architect
  - 4GL Development

If your Sonic Domain Manager is not running during your OpenEdge install, the OpenEdge containers cannot be installed into the Sonic domain, and the following error appears during the installation:

![OpenEdge Adapter for Sonic ESB Information](image)

After your OpenEdge installation completes, start your Sonic Domain Manager, and execute the script indicated in the error message.
Configuration

Based on the type of installation identified, the following configuration takes place in your Sonic environment:

- **Domain installation:**
  - Installs the default service instance, `dev.OpenEdge`, for the OpenEdge Adapter for ESB; this instance is configured to connect directly to the local `esbbroker1` AppServer instance in your OpenEdge install
  - Installs the property file for the OpenEdge Adapter for ESB
  - Seeds the Directory Service for the Resource Editor
  - Installs the OpenEdge `.jar` files for the OpenEdge Adapter for ESB into the Sonic Directory Service.
  - Configures the Sonic Management Console

- **Deployment installation:**
  - Performs all configuration tasks identified for a Domain install
  - Creates a deployment container, named `hostnameContainer`, configured to run the OpenEdge Adapter for ESB and add it to the Management Framework (MF) container

- **Development installation:**
  - Performs all configuration tasks identified for a Deployment install
  - Configures the development container, `dev_OpenEdgeTest`, in the Sonic Workbench

When installation and configuration is complete, the Sonic Management Console has two OpenEdge services under **ESB Configured Objects**:

- **OpenEdge Native Services** — Supports the OpenEdge Adapter for ESB Native Invocation methodology
- **OpenEdge Web Services** — Supports the OpenEdge Adapter for ESB SOAP-based Web Service methodology
Confirming the Sonic Management Console is configured properly (Optional)

Use the Sonic Management Console (SMC) to check the configuration.

To confirm the Sonic Management Console is configured properly:

1. If necessary, start the Sonic Domain Manager and then start the Sonic Management Console.

2. In the left pane of the SMC window, expand the Services folder.

3. Verify OpenEdge Native Services and OpenEdgeWeb Services, as shown in Figure 16–1 and Figure 16–2.

Figure 16–1: OpenEdge Native Services
4. If the OpenEdge Native Services and OpenEdge Web Services are not configured, consult the OpenEdge Adapter for Sonic ESB tailoring installation log file for possible errors. The file is `OpenEdge-install-dir\install\tlr\esbadaptertlr.log`. Successful installation produces the following message at the end of the file:

```
Successfully tailored the OpenEdge SonicESB Adapter
SonicESB Tailoring complete
```

Otherwise errors are listed in the file.
Using the OpenEdge Adapter for Sonic ESB

In the Sonic environment, you use the facilities of the OpenEdge Adapter for Sonic ESB to edit properties and create instances as discussed in the following sections:

- Editing OpenEdge service properties
- Creating an OpenEdge service instance
- Editing an instance of an OpenEdge service

Editing OpenEdge service properties

Default values for the properties of an OpenEdge service are stored in a Sonic ESB resource that is loaded from one of the following files:

- **ESBOE file** — The invocation file associated with the Native Invocation Methodology. Create ESBOE files while you are developing your ABL application by including source code annotations and generating the ESBOE file with OpenEdge Architect or the ESBOEGEN utility. Or you generate the ESBOE files after your development is complete by using ProxyGen.

- **WSM file** — Is a Web Services Mapping (WSM) file, associated with the Web Service Invocation methodology. Use ProxyGen after developing your ABL application, to create a WSM file.

- **WSD file** — Is a Web Service Definition (WSD) file. Use a WSD file if you intend to deploy an existing WSA-based Web service in Sonic ESB. You can export the service definition from OpenEdge Explorer and use the resulting Web Service Definition (WSD) file instead of a WSM file. By doing so, you preserve the defined properties of the service and avoid the need to modify them again.

**Note:** For more information on ProxyGen, see *OpenEdge Development: Open Client Introduction and Programming*.

If you want to change the defaults, follow the instructions provided in the following sections:

- Adding the WSM or WSD resource to the Sonic ESB Directory Service
- Editing the default service properties
- Generating a WSDL file from the OpenEdge Resource Editor

For any specific service instance, you can override some of the defaults stored in the file by editing the desired values in the SMC. You can specify these service-specific values when you create the service instance, and you can edit them after the instance has been defined (see the “Creating an OpenEdge service instance” section on page 16–10). In cases where no value is explicitly defined for a service, the default value is in effect.
By overriding the defaults with service-specific values, you can use the same WSM or WSD resource as the basis for multiple services that differ only with respect to a few details. For example, you might want to create two or more services that provide the same functionality but use the facilities of different AppServers.

Note: Editing the service properties in the SMC does not affect any values stored in the WSM or WSD resource. Conversely, editing the resource does not affect any overrides that were entered for the service instance.

Adding the WSM or WSD resource to the Sonic ESB Directory Service

To make the WSM or WSD file available for use with the Web Service Invocation methodology, the file must be loaded as a resource into the Sonic ESB Directory Service. ProxyGen does this for you when you create a Sonic Web Service Invocation. Follow the steps below to manually load a resource:

To load a WSM or WSD resource:

1. If necessary, start the SonicMQ Domain Manager container, and then start the Sonic Management Console (SMC).
2. Select the Configure tab, then click on the Resources folder.
3. Click OpenEdge Services.
4. Right-click and select Import File. The Select Import File dialog appears. Browse to the location of your WSM or WSD file, and click Import. The right pane of the SMC now shows the name of the file.
5. You can change the name of the resource, if you wish, to any name of your choice that is not already in use (but keep the .wsm or .wsd extension). Then click OK to store the WSM or WSD file in the Directory Service.

Editing the default service properties

Use this procedure to set default properties that will be in effect for services based on a given WSM or WSD file used with the Web Service Invocation methodology. The values that you specify will apply unless they are overridden for specific service instances.

To edit service defaults:

1. Start the Sonic Management Console.
2. Select the Configure tab, then click on the Resources folder.
3. Click OpenEdge Services.
4. Right-click on the WSM file you want to edit, then select Open from the pop-up menu. The Edit OpenEdge Service Definition window appears:

5. The Deployment Information tab includes the following fields:

   - **Web Service Namespace** — This namespace uniquely identifies the service and its elements within Sonic ESB. It must meet the requirements of an XML namespace value. (This default can be overridden by a service-specific value.)

   - **SOAP Action URI** — This is an optional value; if specified, it can be any string. If you enter a value, any client accessing the service as a Web service must place that value in the SOAPAction HTTP header when it invokes operations on the service.

Enter or modify these values as appropriate. Then click the Runtime Properties tab.

6. The Runtime Properties tab shows the values of various properties that affect execution of the service:
If the service was designed to use the session-managed session model, more properties are listed than are shown in the illustration above. For an explanation of each property, see Appendix A, “Reference to OpenEdge Web Service Properties.”

The following properties can be overridden by service-specific values entered in the SMC:

- appServiceProtocol
- appServiceHost
- appServicePort
- appServiceName
- noSessionReuse
- noHostVerify

After setting the properties as appropriate, click the Deployment Information tab if you want to generate a WSDL file, or click OK to save your changes.

Creating an OpenEdge service instance

ProxyGen creates an OpenEdge service instance for you. The following procedure describes how to define an instance of the OpenEdge service type outside of ProxyGen.

**Note:** Before you define an instance of the OpenEdge service type, you must add the WSM or WSD file to the SonicFS for the Web Service Invocation methodology, as described in the “Adding the WSM or WSD resource to the Sonic ESB Directory Service” section on page 16–8.

To create an OpenEdge service instance:

1. If necessary, start the SonicMQ Domain Manager container and then start the Sonic Management Console.

2. Select the Configure tab, then open the Services folder.

3. Click the appropriate OpenEdge Service type: OpenEdge Native Services for a Native Invocation service, or OpenEdge Web Services for a Web Service Invocation.

4. In the right pane, click New. The custom form for specifying service attributes goes into editable mode.

5. The first six fields, in Service Maintenance, specify the properties common to all Sonic ESB services. Only the Service Name is required. Enter a unique name in the Service Name field.

6. Optionally, you can specify an entry endpoint, an exit endpoint, a fault endpoint, and a rejected message endpoint for the service. Service endpoints function as logical connections between services and are used as routing mechanisms. An endpoint can be a SonicMQ queue or topic, a service, or a process. See Sonic ESB Developer’s Guide for more information about endpoints.
You can specify an existing endpoint or create a new one for use as any of the four endpoint values. Click the ellipsis (...) button to the right of an endpoint field (for example, **Entry Endpoint**) to display the **Select Entry Endpoint** window, as shown:

Do one of the following:

- Select an existing endpoint from the list.
- Click **New**. Select **Endpoint**, **Service**, or **Process**; then select the desired option from the submenu to display the **Configure Endpoint** window. Enter the appropriate values and click **OK** to return to the **Select Endpoint** window.

At the **Select Entry Endpoint** window, click **OK**.

7. **Repeat Step 6** to specify the remaining endpoints.

8. Optionally specify a WSDL file to be associated with the OpenEdge service. This is typically the WSDL file generated by means of the OpenEdge resource editor. For more information on generating a WSDL file, see the “Generating a WSDL file from the OpenEdge Resource Editor” section on page 16–13.

To specify a WSDL file, it must already be stored in the **sonicfs** directory. Click the ellipsis (...) button. The **Choose WSDL File Resource** window appears:

Browse to select the WSDL file. Choose **Open**. The WSDL is now associated with your service.
9. The remaining fields, in **Init Parameters**, are specific to the type of OpenEdge service you are creating:

   a. For an OpenEdge Native service, the following parameters are mandatory:
      
      - **AppServer Operating Mode** — Specifies the state of the AppServer
      - **AppServer URL** — Specifies the URL of the AppServer called to execute the ABL code for the service

   Optionally, you can modify the runtime properties, as shown:

   ![Edit OpenEdge Service Runtime Properties](image)

   b. For an OpenEdgeWeb Service, only the **OpenEdge Service Definition File** field requires that you supply a value. You must specify a valid WSM or WSD file. Click the ellipsis (…) button to the right of this field, and browse to the file.

   **Note:** You must have already loaded the file into the SonicFS file system as described in the “Adding the WSM or WSD resource to the Sonic ESB Directory Service” section on page 16–8.

The value in the **SOAP Fault Processing** field determines the action taken when the AppServer returns a SOAP fault message. SOAP faults result from conditions such as the server not running or a message being improperly formatted. They also result form using a RETURN ERROR string in your service. You can ensure that you do not get a SOAP fault from an application error by defining another way to return errors from the service (for example, by using an output parameter).

Choose one of the following three values from the drop-down menu that appears when you click the down arrow at the right of the field:

- **None** — No fault processing occurs. As is always the case with any response from the AppServer, the SOAP fault message is simply sent to the service’s exit endpoint, if any, or to the next step specified in the process itinerary.

- **Message** — The message that was being processed when the SOAP fault was generated is sent to the service’s fault endpoint or to the fault endpoint specified in the process definition. The message can be processed after the fault is corrected.
• **Fault** — The SOAP fault message is sent to the service’s fault endpoint or to the fault endpoint specified in the process definition (perhaps triggering an alert to a person who can take the appropriate corrective action). The message that was being processed when the SOAP fault was generated is lost.

**Note:** If the SOAP Fault Processing field is set to Message or Fault, the service definition or the process definition must specify a fault endpoint. Otherwise, a SOAP fault causes an exception.

Enter values in the remaining fields only if you want to override the default values in the WSM or WSD file specified in the **OpenEdge Service Definition File** field. If values are entered for them, these properties override the defaults as follows:

- **AppServer Protocol** — Overrides the appServiceProtocol default property
- **AppServer Host** — Overrides the appServiceHost default property
- **AppServer Port** — Overrides the appServicePort default property
- **OpenEdge AppService Name** — Overrides the appServiceName default property
- **Disable SSL Hostname Verification** — Overrides the noHostVerify default property
- **Disable SSL Session reuse** — Overrides the noSessionReuse default property

For an explanation of these and other service properties, see Appendix A, “Reference to OpenEdge Web Service Properties.”

10. Click **Apply** to create the service instance.

### Generating a WSDL file from the OpenEdge Resource Editor

You have the option of associating a WSDL file with your service. By generating the correct WSDL file and associating it with your service, you can use the Sonic Web Services Call Composer to generate calls to your service from a workflow.

<table>
<thead>
<tr>
<th>To generate a WSDL file from the Resource Editor:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Open the Sonic Management Console.</td>
</tr>
<tr>
<td>2. Click the <strong>Configure</strong> tab.</td>
</tr>
</tbody>
</table>
3. Right-click the WSM file of your service, then click Open in the pop-up menu. The Edit OpenEdge Service Definition dialog appears:

4. Click Generate WSDL. The Generate WSDL dialog box appears:

5. Specify a Web Service URL that points directly to the service using a sonic: URL address. A sonic: URL address consists of three slash-delimited parts, as shown in the following table:

<table>
<thead>
<tr>
<th>Address part</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sonic:///</td>
<td>sonic:///</td>
<td>Represents the Sonic protocol with no host or port</td>
</tr>
<tr>
<td>node/</td>
<td>local/</td>
<td>Represents the Sonic domain the service is running on, or local for the current domain</td>
</tr>
<tr>
<td>ServiceName</td>
<td>QuoteService</td>
<td>Represents the name of the service</td>
</tr>
</tbody>
</table>

For more information, see Sonic ESB Developer's Guide.

6. In the File Name field, specify a File Name for the WSDL file by either typing the absolute or relative pathname or click Browse, navigate to the directory where you want to create the file, enter a filename, and click Open.

7. Click OK to generate the WSDL file and save it locally.

To associate the WSDL file with your service:

1. At the Sonic Management Console, select the Configure tab.
2. Navigate to the directory where you want to store the WSDL file.
3. Right-click on the directory and select **Import**, as shown:

4. Return to the SMC, then browse to the directory where your WSDL file is saved and select the WSDL file.

   Now your service definition can select the imported WSDL file.

**Editing an instance of an OpenEdge service**

After a service instance has been created, you can edit its properties. To change properties specific to a given service instance, open the **Services** folder in the left pane of the Sonic Management Console and select the desired OpenEdge Service: **OpenEdge Native Services** or **OpenEdge Web Services**. Then select the service instance from the list in the right pane, and set the desired values as explained in the “Creating an OpenEdge service instance” section on page 16–10.
Exposing a service as a standard Web service

By adding a properly configured acceptor to the SonicMQ broker, you can make it possible for a service in Sonic ESB to function as a standard Web service, receiving and responding to requests from clients outside the Sonic ESB environment.

The procedure that follows presents the basic steps for configuring such an acceptor. Refer to Sonic ESB Developer’s Guide for more detailed information.

To add an HTTP/SOAP acceptor to the SonicMQ Broker:

1. Start the SonicMQ management container and then start the SonicMQ Management Console.
2. On the Configure tab, expand the Brokers folder.
3. Select the broker to which you want to add the acceptor (by default, Broker1), and then select Acceptors.
4. Right-click in the right pane to display a pop-up menu. From this menu, select New → HTTP(S) Direct. The New HTTP(S) Direct Acceptor window appears:

5. In the Name field, type a descriptive name.
6. In the first URL field, enter a host name or localhost. In the second (following the colon), enter the port number.
7. Click the **New** button to the right of the **Protocols** list. Then, from the menu that appears, select **HTTP Direct for SOAP**. The **New HTTP Direct for SOAP Protocol** window appears:

![New HTTP Direct for SOAP Protocol](image)

8. In the **Name** field, type a name of your choice. You can use the same name that you gave to the acceptor if you wish.

9. Click **New** and select **Content Reply Send** from the menu. The **New Direct Content Reply Send URL** window appears:

![New Direct Content Reply Send URL](image)

10. Enter the appropriate values at the **New Direct Content Reply Send URL** window and click **OK** at each of the three windows (**New Direct Content Reply Send URL**, **New HTTP Direct for SOAP Protocol**, **New HTTP(S) Direct Acceptor**) to create the acceptor.
Deploying a service instance in Sonic ESB

To be available for use, services must be deployed in an ESB container, and the ESB container must be deployed in a SonicMQ container. You can use existing containers for the deployment of OpenEdge services, or you can create new ones. The procedure below includes steps for creating a new container; skip those steps if you prefer to use existing containers.

**Note:** ProxyGen does this for you when you use it to define a service.

To create an ESB container and deploy the service instance:

1. Start the Sonic Management Console, and on the **Configure** tab select the **ESB Containers** folder.

2. In the right pane, click **New**.

3. In the **ESB Container Maintenance** area of the right pane, type a descriptive string (for example, **OpenEdge_ESB_Container**) in the **Name** field.

4. Click **Apply** to create the container.

5. In the left pane, if necessary, expand the display of the **ESB Containers** folder contents. Then select the container that you want to use.

6. In the right pane, click **New** to display a list of available services and processes. Scroll through the list and select the service that you want to deploy.

7. Click **Apply** to deploy the service in the ESB container.

You can start the new container by creating a boot file and script for it.

To start the new container:

1. Select the new container component (for example, **OpenEdge_Container**) in the **Containers** folder in the left pane of the Sonic Management Console. Right-click and select **Generate Boot File** from the pop-up menu. A file browser appears.

   The boot file is an XML file whose name and location you can specify. Navigate to the desired directory, name the file (for example, **OpenEdge_Container.xml**), and click **Save** to create the file.

2. Finally, you must create a script to start the container. You can use one of the following files as the basis for that script:

   - **Windows** — *SonicMQ-Install-Directory\bin\startcontainer.bat*
   - **UNIX** — *SonicMQ-Install-Directory/bin/startcontainer.sh*

   Make a copy of the appropriate file in a directory of your choice and name it appropriately (for example, **OpenEdge_startcontainer.bat**).
3. Edit the script file and find the following line:

```
set CONTAINERFILE=container.xml
```

Change the value to the full path to the boot file (for example, `OpenEdge_CONTAINER.xml`) that you created in Step 1, and save the file. You can now start the container at any time by executing this script.
Security considerations for OpenEdge Adapter for Sonic ESB

The security of communications between services deployed to the OpenEdge Adapter for Sonic ESB and the clients of those services is a function of two distinct connections, each of which is configured separately with respect to security.

The first connection, that between the OpenEdge Adapter for Sonic ESB and the client, is secured by the facilities of Sonic ESB and thus is outside the scope of OpenEdge administration. See the Sonic ESB documentation for information about making this connection secure.

The second connection is via AppServer protocol between the deployed service and the AppServer. It is recommended that you run your AppServer and Sonic container hosting the service on the same machine to ensure that the AppServer protocol is secure without using single sign-on (SSO).

Otherwise, for this connection to be secure, the following conditions must be met:

- You must obtain and install public key certificates for the OpenEdge Adapter for Sonic ESB host machine.

- The service must send SSL requests to the AppServer that is to process the client requests. To configure the service to send SSL requests, you set the value of the `appServiceProtocol` property to `AppServerS` or `AppServerDCS`. You set this property, either for a specific service (see the “Editing an instance of an OpenEdge service” section on page 16–15) or as the default for services deployed to a given adapter instance (see the “Editing the default service properties” section on page 16–8). Note that this property applies to deployed services, not to the WSA itself.

- The AppServer must be SSL-enabled, meaning that it accepts SSL requests from the OpenEdge Adapter for Sonic ESB (or other clients). You set the property `sslEnable=1` by checking the `Enable SSL client connections` box in the SSL General properties category in OpenEdge Explorer, or by manually editing the `ubroker.properties` file. You must also obtain and install a server private key and public key certificate and set additional SSL server properties. See the “SSL-enabled AppServer operation” section on page 2–8 for more information.

Note: You can use the `mergeprop` utility installed with OpenEdge to manually edit the `ubroker.properties` file. For information on using `mergeprop`, see OpenEdge Getting Started: Installation and Configuration.

For more information on SSL support in OpenEdge, including configuring and operating a Sonic ESB service as a client of an SSL-enabled AppServer, see OpenEdge Getting Started: Core Business Services.
SSL-related service properties

You can set the following properties, either as defaults for services deployed to a given OpenEdge Adapter for Sonic ESB instance or as properties of a specific service:

- **appServiceProtocol** — Assigns a value of AppServerS or AppServerDCS to support SSL communication with the AppServer

- **noHostVerify** — Controls whether the WSA compares the host name of the connecting AppServer with the Common Name specified in the server digital certificate

- **noSessionReuse** — Controls whether the service requests reuse of the session ID for successive connections to the same AppServer

For more information about these and other service properties, see Appendix A, “Reference to OpenEdge Web Service Properties.”
Part VI

Appendixes

Appendix A, Reference to OpenEdge Web Service Properties
Appendix B, Command and Utility Reference
Appendix C, Reference to Dynamic Server Properties
Reference to OpenEdge Web Service Properties

You can set the properties described in this appendix to affect execution of a deployed Web service. The properties that are available for you to set depend on the session model of the Web service (session managed or session free). You can verify the session model of a Web service by viewing its status.

The properties described in this appendix apply to a OpenEdge Web service whether you deploy it as a Web service using an instance of the Web Services Adapter (WSA) in the OpenEdge environment or you deploy it as an OpenEdge service using the OpenEdge Adapter for Sonic ESB to install it in the Sonic Enterprise Service Bus (ESB) environment. In the Sonic ESB environment an OpenEdge service can function either as a Web service (as in the OpenEdge environment) or in integration with other processes on the Enterprise Service Bus. In all cases, these properties have the same effect on service functionality.

This appendix contains the following sections:

- Overview

- Alphabetical reference
Overview

You can set default values for Web service properties on a WSA instance using OpenEdge Explorer, which then apply to every Web service that you deploy to that WSA instance. Once deployed, you can review and change these values for each individual Web service.

You can set the same properties for an OpenEdge service in the Sonic ESB environment, using the custom resource editor for OpenEdge services accessible from the Sonic ESB Explorer.

In the following sections, service refers to either a OpenEdge Web service or an OpenEdge service, and adapter refers to either the WSA Adapter or the OpenEdge Adapter for Sonic ESB.

Summary of properties

Table A–1 lists a summary of all the properties that you can set for a service, depending on its session model (free, managed, or both).

Table A–1: Service properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Session model</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appServiceHost</td>
<td>Both</td>
<td>Host name for the NameServer or AppServer that supports an application service</td>
</tr>
<tr>
<td>appServiceName</td>
<td>Both</td>
<td>The name of an application service supported by the specified NameServer or AppServer and that supports all of the operations that define the service</td>
</tr>
<tr>
<td>appServicePort</td>
<td>Both</td>
<td>The port number to access the application service bound by the service</td>
</tr>
<tr>
<td>appServiceProtocol</td>
<td>Both</td>
<td>The protocol that the adapter instance uses to access the host for the specified NameServer or AppServer</td>
</tr>
<tr>
<td>connectionLifetime</td>
<td>Free</td>
<td>The maximum lifetime (in seconds) of AppServer connections in the connection pool for this service</td>
</tr>
<tr>
<td>idleSessionTimeout</td>
<td>Free</td>
<td>The duration (in seconds) between attempts by the adapter to shut down extra network connections to the AppServer, based on client demand</td>
</tr>
<tr>
<td>initialSessions</td>
<td>Free</td>
<td>The number of network sessions to be created (and shared by all clients) when the connection pool for the service is initialized by the adapter</td>
</tr>
<tr>
<td>maxSessions</td>
<td>Free</td>
<td>The maximum number of connected sessions allowed in the service connection pool</td>
</tr>
<tr>
<td>minIdleConnections</td>
<td>Free</td>
<td>The minimum number of idle, or inactive, AppServer connections to maintain</td>
</tr>
<tr>
<td>Property</td>
<td>Session model</td>
<td>Short description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>minSessions</td>
<td>Free</td>
<td>The minimum number of connected sessions that the adapter attempts to maintain in the service connection pool</td>
</tr>
<tr>
<td>noHostVerify</td>
<td>Both</td>
<td>Controls whether the service verifies that the host name of the connecting AppServer matches the Common Name specified in the server digital certificate (meaningful only in the context of SSL connections)</td>
</tr>
<tr>
<td>noSessionReuse</td>
<td>Both</td>
<td>Controls whether the service requests reuse of the SSL session ID when reconnecting to the same AppServer (meaningful only in the context of SSL connections)</td>
</tr>
<tr>
<td>nsClientMaxPort</td>
<td>Both</td>
<td>The maximum value for the adapter to specify for the port number used to communicate with a NameServer</td>
</tr>
<tr>
<td>nsClientMinPort</td>
<td>Both</td>
<td>The minimum value for the adapter to specify for the port number used to communicate with a NameServer</td>
</tr>
<tr>
<td>nsClientPicklistExpiration</td>
<td>Free</td>
<td>The maximum duration (in seconds) that the adapter retains a list of AppServer options (pick list) for an idle application service</td>
</tr>
<tr>
<td>nsClientPicklistSize</td>
<td>Free</td>
<td>The number of available AppServer options (the broker pick list) that the adapter requests from a NameServer each time it looks up a given application service name</td>
</tr>
<tr>
<td>nsClientPortRetry</td>
<td>Both</td>
<td>The maximum number of requests that the adapter makes for a valid local port number when attempting to communicate with a NameServer</td>
</tr>
<tr>
<td>nsClientPortRetryInterval</td>
<td>Both</td>
<td>The interval (in milliseconds) that the adapter waits between requests to get a valid port number when attempting to communicate with a NameServer</td>
</tr>
<tr>
<td>requestWaitTimeout</td>
<td>Free</td>
<td>Determines how the adapter handles requests when the service connection pool becomes full</td>
</tr>
<tr>
<td>serviceFaultLevel</td>
<td>Both</td>
<td>The amount of information returned to the client for a SOAP Fault</td>
</tr>
<tr>
<td>serviceLoggingLevel</td>
<td>Both</td>
<td>The amount and type of information written to the service log by the adapter for each log entry</td>
</tr>
</tbody>
</table>
A connection pool is a cache that the adapter manages for each session-free service that relies on a controlling NameServer to access the AppServer that provides the corresponding application service. This cache maintains one or more AppServer connections for the application service, which are made available by the NameServer and otherwise managed through these properties. For more information on connection pools, see Chapter 6, “Deploying and Managing OpenEdge Web Services.”

### Setting Service properties

You must disable a service before you can set most properties on it. You can verify if the service is disabled by viewing its status. The only properties you can set for an enabled service are:

- `serviceFaultLevel`
- `serviceLoggingLevel`

If you set other properties while the service is enabled, the property value changes take effect only after you disable, then enable the service again.
Alphabetical reference

This section contains an alphabetical reference to the properties that you can set for a service, including the applicable session model, a complete description of usage, and the default value for each property.

**appServiceHost**

**Session model**: Managed or Free

Host name for the NameServer or AppServer that supports an application service with the name specified by the `appServiceName` property.

**Installation default**: localhost

**appServiceName**

**Session model**: Managed or Free

The name of an application service supported by the NameServer or AppServer specified by the `appServiceHost` property, and that supports all operations that define the service.

**Installation default**: asbroker1

**appServicePort**

**Session model**: Managed or Free

The UDP (NameServer) or TCP (AppServer) port number to access the application service supported by the host specified by the `appServiceHost` property.

**Installation default**: 5162

**appServiceProtocol**

**Session model**: Managed or Free

The protocol that the WSA Adapter or OpenEdge Adapter for Sonic ESB uses to access the host specified by the `appServiceHost` property. For a NameServer host, this is “AppServer”; for an AppServer host, “AppServerDC”.

**Installation default**: AppServer
connectionLifetime

Session model: Free

The maximum lifetime (in seconds) of AppServer connections in the connection pool for this service. A value of 0 indicates that the lifetime of these connections is unlimited, unless they are disconnected according to the requirements of other property settings, such as idleSessionTimeout. A positive value indicates that the WSA Adapter or OpenEdge Adapter for Sonic ESB maintains any AppServer connections for this service for the specified number of seconds. Thus, when the idleSessionTimeout interval expires, the adapter does any necessary trimming of connections in the connection pool beginning with those whose connectionLifetime interval has expired. However, the adapter maintains the connections for all services whose connectionLifetime interval has not yet expired regardless of other property settings.

Installation default: 0

idleSessionTimeout

Session model: Free only

The duration (in seconds) between attempts by the WSA Adapter or OpenEdge Adapter for Sonic ESB to shut down extra network connections to the AppServer, based on client demand. The adapter does this by monitoring the maximum number of sessions needed since the last time-out, then disconnecting any connections in excess of that number.

A value of 0 indicates that the adapter will never disconnect idle sessions unless the connectionLifetime interval has expired.

Installation default: 0

initialSessions

Session model: Free only

The number of network sessions to be created (and shared by all clients) when the connection pool for the service is initialized by the WSA Adapter or OpenEdge Adapter for Sonic ESB. This value must be between the value of the minSessions property and the maxSessions property, inclusive, unless maxSessions is set to 0. If maxSessions is set to 0, the initialSessions value must only be greater than or equal to minSessions.

Installation default: 1

maxSessions

Session model: Free only

The maximum number of connected sessions allowed in the connection pool. Once the number of sessions in the pool reaches the limit specified by maxSessions, the WSA Adapter or OpenEdge Adapter for Sonic ESB creates no additional sessions for this service, and handles all requests for this service according to the requestWaitTimeout property setting.

A value of 0 indicates that the size of the connection pool is unlimited.

Installation default: 0
**minIdleConnections**

**Session model:** Free only

The minimum number of idle, or inactive, AppServer connections to maintain. Given sufficient AppServer resources, this setting allows the WSA Adapter or OpenEdge Adapter for Sonic ESB to always have a free connection available when a new request arrives, so the request does not have to wait for the adapter to locate and establish a new connection to service it. When the value of this property is greater than zero, the adapter uses its free time to pre-allocate network connections for the service connection pool.

**Installation default:** 0

**minSessions**

**Session model:** Free only

The minimum number of connected sessions allowed in the connection pool. The WSA Adapter or OpenEdge Adapter for Sonic ESB attempts to keep at least this many sessions connected to the application service (AppServer).

**Installation default:** 1

**noHostVerify**

**Session model:** Both

If set to 1, turns off host verification for an SSL Web service connection (specified by the `appServiceProtocol` property). If cleared, the WSA compares the host name of the connecting AppServer with the Common Name specified in the server digital certificate, and raises a Web service error if they do not match. With this parameter specified, the Web service never raises the error.

**Installation default:** 0

**noSessionReuse**

**Session model:** Both

If set to 1, the Web service connection does not reuse the SSL session ID when reconnecting to the same AppServer for an SSL Web service connection (specified by the `appServiceProtocol` property).

**Installation default:** 0
nsClientMaxPort

Session model: Managed or Free

The maximum value for the UDP port number that the WSA Adapter or OpenEdge Adapter for Sonic ESB specifies when communicating with the NameServer. This value must be greater than or equal to the value of the nsClientMinPort property.

This property applies only to services that use a NameServer to access application services (an AppServer).

Installation default: 0

nsClientMinPort

Session model: Managed or Free

The minimum value for the WSA Adapter or OpenEdge Adapter for Sonic ESB to specify for the UDP port number when communicating with the NameServer. The value must be less than or equal to the value of the nsClientMaxPort property.

If this value is 0, the adapter chooses the NameServer client port number randomly.

This property applies only to services that use a NameServer to access application services (an AppServer).

Installation default: 0

nsClientPicklistExpiration

Session model: Free only

The maximum duration (in seconds) that the WSA Adapter or OpenEdge Adapter for Sonic ESB retains a list of AppServer options (pick list) for an idle application service.

A value of 0 indicates that the pick list never expires.

This property applies only to services that use a NameServer to access application services (an AppServer).

Installation default: 0

nsClientPicklistSize

Session model: Free only

The number of available AppServer options (the broker pick list) that the WSA Adapter or OpenEdge Adapter for Sonic ESB requests from the NameServer each time it looks up a given application service name.

This property applies only to services that use a NameServer to access application services (an AppServer).

Installation default: 1
nsClientPortRetry

Session model: Managed or Free

The maximum number of requests that the WSA Adapter or OpenEdge Adapter for Sonic ESB makes for a valid local UDP port number when attempting to communicate with the NameServer on behalf of the service.

This property applies only to services that use a NameServer to access application services (an AppServer).

Installation default: 3

nsClientPortRetryInterval

Session model: Managed or Free

The interval (in milliseconds) that the WSA Adapter or OpenEdge Adapter for Sonic ESB waits between requests to get a valid UDP port number when attempting to communicate with the NameServer on behalf of the service.

This property applies only to services that use a NameServer to access application services (an AppServer).

Installation default: 200

requestWaitTimeout

Session model: Free

Determines how the WSA Adapter or OpenEdge Adapter for Sonic ESB handles requests when the connection pool becomes full. (The connection pool is full when the number of active sessions equals the value of maxSessions, and all sessions are currently running requests.)

The adapter handles such requests according to the value of requestWaitTimeout, as shown in Table A–2.

Table A–2: requestWaitTimeout property values

<table>
<thead>
<tr>
<th>If the value is . . .</th>
<th>The WSA Adapter or OpenEdge Adapter for Sonic ESB . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Queues the request indefinitely until an AppServer session becomes available.</td>
</tr>
<tr>
<td>0</td>
<td>Rejects the request and returns an error message to the client indicating that there are too many concurrent requests.</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>Queues the request for the maximum number of seconds specified by the value until an AppServer session becomes available. If no session becomes available in that time, the adapter returns an error to the client.</td>
</tr>
</tbody>
</table>

Installation default: 0
**serviceFaultLevel**

**Session model:** Managed or Free

The amount (level) of information returned to the client for a SOAP Fault as determined by an integer value. A level of 2 returns basic information in the `<FaultCode>` and `<FaultString>` elements for each SOAP Fault message, which is suitable for normal production environments. A level of 3 returns more detailed information that is suitable for development environments. Other values provide varying levels of diagnostic information, and are reserved for use by Progress Technical Support and Engineering.

**Installation default:** 2

**serviceLoggingLevel**

**Session model:** Managed or Free

The amount and type of information written for each service log entry by the WSA Adapter or OpenEdge Adapter for Sonic ESB to the adapter log file, cumulatively determined by the integer values from 1 to 4, as shown in Table A–3.

<table>
<thead>
<tr>
<th>If the value is . . .</th>
<th>The WSA Adapter or OpenEdge Adapter for Sonic ESB logs . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Errors internally detected by the adapter only (errors that result in SOAP Faults not included)</td>
</tr>
<tr>
<td>2</td>
<td>Additional errors that result in SOAP Faults returned to the client</td>
</tr>
<tr>
<td>3</td>
<td>Additional debugging information that is helpful to the deployer</td>
</tr>
<tr>
<td>4</td>
<td>Additional debugging information that is helpful to Progress support services and engineering</td>
</tr>
</tbody>
</table>

**Installation default:** 2

**staleO4GLObjectTimeout**

**Session model:** Managed or Free

The maximum duration (in seconds) that a service object (AppObject, SubAppObject, or ProcObject) can be idle before it is released.

As part of managing certain service objects with OpenEdge, clients explicitly create them using factory methods before invoking other methods on them. When the client no longer requires the object, it has the responsibility to release the object from the service run-time context. However, if this time-out expires before the client releases the object, the WSA Adapter or OpenEdge Adapter for Sonic ESB assumes that the client application no longer requires access to the object, and deletes it from the service run-time context automatically. In effect, the adapter uses this time-out to provide garbage collection on service objects that client applications stop referencing and fail to release in the specified period of time. Any subsequent attempt by a client to access this object returns an error from the adapter.
A value of 0 for this property specifies that the adapter perform no garbage collection on idle objects, leaving the responsibility for releasing them entirely to client applications.

When all objects that reference the connection pool have been released, the adapter also releases the now unreferenced connection pool as well.

**Installation default:** 0

### waitIfBusy

**Session model:** Managed only

An integer value that determines how to handle client requests to a service that is busy processing a prior request. If the value is 1, the WSA Adapter or OpenEdge Adapter for Sonic ESB queues multiple requests for this service and executes them one at a time until the queue is empty. If the value is 0 and the adapter is executing a prior request for the service, each subsequent request for the same service fails until the adapter completes the request it is currently executing.

**Installation default:** 0
This appendix describes the following commands and utilities that you use to configure, manage, start, and stop AdminServer, Application Internet Adapter (AIA), AppServer, OpenEdge Adapter for SonicMQ BrokerConnect, Web Services Adapter (WSA), WebSpeed, and the generation of OpenEdge Adapter for Sonic ESB native invocation files:

- ADAPTCONFIG
- ADAPTMAN
- AIACONFIG
- ASBMAN
- ASCONFIG
- ESBOEGEN
- NSCONFIG
- NSMAN
- PROADSV
- WSACONFIG
- WSAMAN deploy
- WSAMAN disable
- WSAMAN enable
- WSAMAN export
- WSAMAN getdefaults
- `WSAMAN getprops (Service)`
- `WSAMAN getprops (WSA)`
- `WSAMAN getstats (Service)`
- `WSAMAN getstats (WSA)`
- `WSAMAN import`
- `WSAMAN list`  
- `WSAMAN query (Service)`
- `WSAMAN query (WSA)`
- `WSAMAN resetdefaults`  
- `WSAMAN resetprops`  
- `WSAMAN resetstats (Service)`
- `WSAMAN resetstats (WSA)`
- `WSAMAN setdefault`  
- `WSAMAN setprops (Service)`
- `WSAMAN setprops (WSA)`
- `WSAMAN undeploy`  
- `WSAMAN update`  
- `WSCONFIG`  
- `WTBMAN`
ADAPTCONFIG

Validates manual changes made to the ubroker.properties file for an OpenEdge Adapter for SonicMQ BrokerConnect instance.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>adaptconfig [</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
</tbody>
</table>

- **-name adapter-broker**

    Name of the SonicMQ Broker for BrokerConnect (required). -i is also valid.

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

    Full properties file path (optional). -f is also valid.

- **-validate**

    Validate. -v is also valid.

- **-help**

    Displays command-line help. -h is also valid.

**Note**

If No options is used, the results lists all defined SonicMQ Brokers for BrokerConnect.
**ADAPTMAN**

Starts, stops, queries, and kills an existing OpenEdge Adapter for SonicMQ BrokerConnect broker or manipulates brokers on other machines by specifying name of the machine and the port the AdminServer is running on.

**Syntax**

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UNIX</strong></td>
<td>adaptman {</td>
</tr>
<tr>
<td></td>
<td>{ -name adapter-broker</td>
</tr>
<tr>
<td></td>
<td>{ -kill</td>
</tr>
<tr>
<td></td>
<td>[ -host host-name -user user-name</td>
</tr>
<tr>
<td></td>
<td>[ -port port-number ]</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td><strong>Windows</strong></td>
<td></td>
</tr>
</tbody>
</table>

- **-name adapter-broker**
  
  Name of the SonicMQ Broker for BrokerConnect (required). `-i` is also valid.

- **-kill**
  
  Causes emergency shutdown of the SonicMQ Broker for BrokerConnect. `-k` is also valid.

- **-start**
  
  Starts the named SonicMQ Broker for BrokerConnect. `-x` is also valid.

- **-stop**
  
  Stops the SonicMQ Broker for BrokerConnect. `-e` is also valid.

- **-query**
  
  Queries the named SonicMQ Broker for BrokerConnect. `-q` is also valid.

- **-listallprops**
  
  Displays all active broker properties, including updated values for dynamic properties that have changed.

- **-host host-name**
  
  Host name where the AdminServer is running.

- **-user user-name**
  
  User name. `-u` is also valid.

- **-port port-number**
  
  Port number of the running AdminServer.

- **-help**
  
  Displays command-line help. `-h` is also valid.
Note

Enter the -i or the -name parameter followed by the name of the SonicMQ Broker for BrokerConnect and then the command to start, stop, query, or kill a broker.

Examples

Table B–1 shows several examples that use the adaptman command to start an instance called SonicMQ1.

<table>
<thead>
<tr>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start an instance called SonicMQ1</td>
<td>adaptman -i sonicMQ1 -start</td>
</tr>
<tr>
<td>Query the instance for its status</td>
<td>adaptman -i sonicMQ1 -query</td>
</tr>
<tr>
<td>Stop an instance</td>
<td>adaptman -name sonicMQ1 -stop</td>
</tr>
<tr>
<td>Get status of an instance on the machine</td>
<td>adaptman -host xxxxxx -port 12935 -i sonicMQ1 -q</td>
</tr>
<tr>
<td>whose AdminServer is on port 12935</td>
<td></td>
</tr>
<tr>
<td>Kill an instance</td>
<td>adaptman -i sonicMQ1 -kill</td>
</tr>
</tbody>
</table>
AIACONFIG

Validates changes made to the ubroker.properties file for an AIA instance.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX Windows</td>
<td>aiaconfig [ [ [ -name AIA-instance-name ] [ -propfile path-to-properties-file ] [ -validate ] ] [ -help ] ]</td>
</tr>
</tbody>
</table>

-**name AIA-instance-name**

   Name of the AIA instance (required).

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

   Full properties file path (optional). -f is also valid.

-**validate**

   Validate. -v is also valid.

-**help**

   Displays command-line help. -h is also valid.

**Note**

If you upgrade to a new version of OpenEdge, you might want to retain changes made to the previous version's ubroker.properties file. If that is the case, place the old properties file in the installed properties directory to replace the default. When starting the AdminServer for the first time, a properties file conversion utility automatically runs.
ASBMAN


Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| UNIX             | asbman \{ 
|                  | { -name AppServer-name 
|                  | { -kill | -start | -stop | -query | 
|                  |   -addservers number-to-start | 
|                  |   -trimservers number-to-trim | 
|                  |   -listclients | 
|                  |   -clientdetail connection-handle | 
|                  |   -listallprops } 
|                  | } [ -host host-name -user user-name | -user user-name ] 
|                  | [ -port port-number ] 
|                  | } | -help } |
| Windows          |        |

-name AppServer-name

This parameter is required. It specifies the name of an AppServer.

-kill

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

-start

Starts an AppServer.

-stop

Tells the AppServer to stop itself.

Note: The AppServer stops only after completing any active client requests.

-query

Queries an AppServer for its status.

-addservers number-to-start

Specifies the number of additional servers to start.

-trimservers number-to-trim

Specifies the number of additional servers to trim.
-listclients

Provides a tabular summary of all clients connected to the AppServer along with information about the clients. Table B–2 describes the information provided by the command.

Table B–2: Summary view output fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnHdl (connection handle)</td>
<td>A unique value that identifies the connection. This value is a monotonically increasing number that is assigned when the client connects to the AppServer.</td>
</tr>
<tr>
<td>User (username)</td>
<td>The string passed as the user name parameter in the AppServer CONNECT method; otherwise blank. Interpretation of this value is dependent on the application.</td>
</tr>
<tr>
<td>Rmt IP (remote IP address)</td>
<td>The IP address of the host machine where the client resides.</td>
</tr>
<tr>
<td>Rmt Port (remote port number)</td>
<td>The port number of the client on the client host machine.</td>
</tr>
</tbody>
</table>
| State (connection state) | A string that identifies the state of the connection at the time the query was performed. Possible values are:  
  - CONNECTING  
  - CONNECTED  
  - SENDING  
  - RECEIVING  
  - DISCONNECTING |

Here is sample output for an AppServer with four connections, three of which connected with user names specified:

<table>
<thead>
<tr>
<th>ConnHdl</th>
<th>User</th>
<th>Rmt IP</th>
<th>Rmt Port</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1002</td>
<td>jsmith</td>
<td>fd00:19d:808e:1::30</td>
<td>33445</td>
<td>RECEIVING</td>
</tr>
<tr>
<td>1003</td>
<td>ljones</td>
<td>127.0.0.1</td>
<td>33457</td>
<td>CONNECTED</td>
</tr>
<tr>
<td>1005</td>
<td>fe80::211:43ff:fe37:f598</td>
<td>3546</td>
<td>SENDING</td>
<td></td>
</tr>
<tr>
<td>1009</td>
<td>msardy</td>
<td>172.168.0.100</td>
<td>13457</td>
<td>CONNECTING</td>
</tr>
</tbody>
</table>
-clientdetail connection-handle

Provides a detailed view of a specific client that is currently connected to the AppServer. Use -listclients first to find the connection-handle value. The detailed view displays all the information shown in the summary view (-listclients) plus the information described in Table B–3.

Table B–3: Additional detail view output fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection ID</td>
<td>The globally unique identifier assigned to each client connection at the time the client connects to the AppServer.</td>
</tr>
<tr>
<td></td>
<td>This is usually the same value that is accessible to the ABL client application using the CLIENT-CONNECTION-ID attribute on the server object handle, and to the ABL server application using the SERVER-CONNECTION-ID attribute on the session handle.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> In state-free operating mode, this value may not be the same as the CLIENT-CONNECTION-ID of the server object handle. In state-free mode, the server object handle is a pool of physical connections, each with a unique client connection ID. Therefore, in state-free operating mode, the CLIENT-CONNECTION-ID of the server object handle is the client connection ID of the first connection added to the pool of connections.</td>
</tr>
<tr>
<td>request count</td>
<td>The number of requests executed by the client on the connection including the connection request itself.</td>
</tr>
<tr>
<td>agent PID</td>
<td>The process identifier of the AppServer agent actively servicing a request from the specified client. Blank if no request is active.</td>
</tr>
<tr>
<td>agent port number</td>
<td>The listening port number of the AppServer agent actively servicing a request from the specified client. Blank if no request is active.</td>
</tr>
</tbody>
</table>

Here is sample output:

```
connection handle= 1002
username= moe1024
remote IP address= fd00:19d:808e:1::30
remote port number= 33445
connection state= SENDING
connection ID= fd00:19d:808e:1::30::ub1::30901::218a44e2518a4557:3be3e697:11866ab798e:-7ffd
request count= 5001
agent PID= 24336
agent port number= 2006
```
-listallprops

Displays all active broker properties, including updated values for dynamic properties that have changed.

-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. 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.

Windows supports three different formats for user-name:

- A user name as a simple text string, such as "mary", implies a local user whose user account is defined on the local Windows server machine, 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 Windows 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 Windows Server, including the one where the AdminServer is running.

-port port-number

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

-help

Displays command-line help.
Examples Table B–4 shows several examples that use the ASBMAN command. Assume the AppServer instance is AS1 and the NameServer is NS1.

Table B–4: ASBMAN command examples

<table>
<thead>
<tr>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
</table>
| Start a local AppServer instance after starting the local controlling NameServer. | nsman -name NS1 -start  
asbman -name AS1 -start |
| Start a remote AppServer instance after starting a remote controlling NameServer.¹ | nsman -name NS1 -host nsserve -port 20950 -user daniel -start  
asbman -name AS1 -host asserve -port 20950 -user daniel -start |
| Stop a local AppServer instance (AS1) and its controlling NameServer instance (NS1). | asbman -name AS1 -stop  
nsman -name NS1 -stop |

¹. The AppServer and controlling NameServer are on different hosts and happen to use the same TCP/IP port number to access the AdminServer on each host. If you specify a host, OpenEdge Explorer always prompts for a user name (if necessary) and password. In this example, the commands specify the user name and prompt only for the password.
ASCONFIG

Displays the property settings associated with an AppServer configuration and checks that the syntax and values in the ubroker.properties file are valid. You must run the ASCONFIG utility locally on the machine on which the AppServer is running. The utility does not run across the network.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| UNIX             | asconfig [ [ -name AppServer-name ] [ -propfile path-to-properties-file ] -validate ] | -name AppServer-name
| Windows          | asconfig [ -help ] |

- **-name AppServer-name**

  Specifies which existing AppServer configuration to examine. The name must match the name of an existing AppServer configuration in the specified properties file. If you do not specify an AppServer, the ASCONFIG utility analyzes all AppServer 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, such as:
  - %DLC%\properties\ubroker.properties in Windows.
  - $DLC/properties/ubroker.properties on UNIX.

- **-validate**

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

- **-help**

  Displays command-line help.

**Note**

Never needed if you always use OpenEdge Explorer.

**Example**

The following command validates the syntax and views the configurations of all AppServer instances defined within the test.properties file located in the current working directory:

```
asconfig -propfile test.properties -validate
```
ESBOEGEN

ESBOEGEN processes annotated ABL source files to generate .esboe files.

Syntax

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

- **-source directory**
  Specify the directory containing annotated ABL files.

- **-esboe directory**
  Specify the destination directory for generated .esboe files.

- **-archive filename**
  Specify the name of an archive (.xar) file to hold all the generated .esboe files. `filename` must be a fully qualified filename.

- **-rcode directory**
  Specify the directory containing the compiled r-code that corresponds to either the directory specified with `-source` or the listed files. This parameter is required to process ABL code if it contains temp-table definitions containing the keyword LIKE.

- **-recurse**
  Direct ESBOEGEN to recursively search all subdirectories for ABL code. When specified, a corresponding directory tree is built in the output directory specified with `-esboe` or in the archive specified with `-archive`.

- **files**
  A comma separated list of ABL files. If a fully qualified file name is not specified, then ESBOEGEN looks for the file in the current working directory.
Notes

• OpenEdge Architect is required to run ESBOEGEN, and is only supported in Windows.

• You cannot specify both -esboe and -archive.

• ESBOEGEN selects the output destination of the generated .esboe files to according to the following order of precedence:
  – Destination specified by -esboe or -archive
  – Directory specified by -rcode
  – Directory specified -source
  – Directory specified for file name listed with files
Displays the property settings associated with a NameServer configuration and checks that the syntax and values are valid. The NSCONFIG utility runs locally on the machine where the AdminService is running. The utility does not run across the network.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td><code>nsconfig [ [ [ -name name-server ] [ -propfile path-to-properties-file ] [ -validate ] ] [ -help ] ]</code></td>
</tr>
</tbody>
</table>

- **-name name-server**
  Specifies which existing NameServer configuration to examine. The name must match the name of an existing NameServer configuration in the specified properties file. If you do not specify a NameServer, the NSCONFIG utility analyzes all NameServer 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, such as:
  - `install-path\properties\ubroker.properties` in Windows
  - `install-path/properties/ubroker.properties` on UNIX

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

- **-help**
  Displays command-line help.

Notes

- Never needed if you always use OpenEdge Explorer.
- A single NameServer can simultaneously support all of the AppServer, WebSpeed, and DataServer products using OpenEdge Explorer.
Table B–5 shows two examples that use the NSCONFIG command. Assume the NameServer is NS1.

Table B–5: NSCONFIG command examples

<table>
<thead>
<tr>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>View a NameServer configuration</td>
<td>nsconfig -name NS1</td>
</tr>
<tr>
<td>View a NameServer configuration defined within a different property file</td>
<td>nsconfig -name NS1 -propfile g:\other.properties -validate</td>
</tr>
</tbody>
</table>

For information on managing a NameServer using the NSCONFIG utility, see *OpenEdge Getting Started: Installation and Configuration.*
Controls the operation of a configured NameServer. The utility allows you to start a NameServer, query its status, and shut down a NameServer.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>nsman {</td>
</tr>
<tr>
<td></td>
<td>-name nameserver</td>
</tr>
<tr>
<td></td>
<td>{ -kill</td>
</tr>
<tr>
<td></td>
<td>[ -host host-name -user user-name</td>
</tr>
<tr>
<td></td>
<td>[ -port port-number ]</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

-name name-server

This parameter is required. It specifies the name of the NameServer.

-kill

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

-start

Starts the NameServer.

-stop

Stops the NameServer.

-query

Queries the NameServer for its status.

-listallprops

Displays all active broker properties, including updated values for dynamic properties that have changed.

-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. 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.

-port port-number

Specifies the port number of the machine where the AdminServer is running. If a port number is not specified, it defaults to 20931.
-help

Displays command-line help.

Notes

• A single NameServer can simultaneously support all of the AppServer, WebSpeed, and DataServer products.

• 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 NT 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 NT Server, including the one where the AdminServer is running.

Examples

Table B–6 shows examples that use the NSMAN command. Assume the NameServer is NS1; the user name is tom; and the AdminServer is on the remote host finance on the port 9999.

Table B–6: NSMAN command examples

<table>
<thead>
<tr>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start a local NameServer</td>
<td>nsman -name NS1 -start</td>
</tr>
<tr>
<td>Start a remote NameServer¹</td>
<td>nsman -name NS1 -host finance -port 9999 -user tom -start</td>
</tr>
<tr>
<td>Query a local NameServer</td>
<td>nsman -name NS1 -query</td>
</tr>
<tr>
<td>Query a remote NameServer¹</td>
<td>nsman -name NS1 -host finance -port 9999 -user tom -query</td>
</tr>
<tr>
<td>Stop a local NameServer</td>
<td>nsman -name NS1 -stop</td>
</tr>
<tr>
<td>Stop a remote NameServer¹</td>
<td>nsman -name NS1 -host finance -port 9999 -user tom -stop</td>
</tr>
</tbody>
</table>

¹. Prompts for a password.

For information on managing a NameServer using the NSMAN utility, see OpenEdge Getting Started: Installation and Configuration.
PROADSV

Starts, stops, and queries the status of the AdminServer on UNIX. In Windows, you start the AdminService as a Windows service using the Services applet in the Control Panel.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>proadsv { -start</td>
</tr>
<tr>
<td></td>
<td>[ -port port-number ]</td>
</tr>
<tr>
<td></td>
<td>[ -adminport port-number ]</td>
</tr>
<tr>
<td></td>
<td>[ -help ]</td>
</tr>
</tbody>
</table>

-`start`

Starts the AdminServer.

-`stop`

Stops the AdminServer.

-`query`

Displays the AdminServer status.

-`port port-number`

Specifies the listening port number. If a port number is not specified, the port defaults to 20931.

-`adminport port-number`

Specifies the port number used by the AdminServer for database broker communication. If a port number is not specified, the `adminport` defaults to port 7832.

-`help`

Displays the command-line help.

Examples

Table B–7 shows several examples that use the NSCONFIG command. Assume the NameServer is NS1.

<table>
<thead>
<tr>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start the AdminServer on UNIX</td>
<td>proadsv -start</td>
</tr>
<tr>
<td>Start the AdminServer on UNIX using a specified port</td>
<td>proadsv -port port-number -start</td>
</tr>
<tr>
<td>Query the AdminServer on UNIX</td>
<td>proadsv -host host-name -user user-name -query</td>
</tr>
</tbody>
</table>

For more information on the PROADSV utility, see OpenEdge Getting Started: Installation and Configuration or OpenEdge Data Management: Database Administration.
**WSACONFIG**

Validates Web Services Adapter configurations. The **WSACONFIG** utility displays the property settings associated with WSA configuration and checks that the syntax and values are valid. You must run the **WSACONFIG** utility locally on the machine where the WSA is running. The utility does not run across the network.

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UNIX</strong></td>
<td><code>wsaconfig</code> [ [ [-name wSA-instance-name] ] [ [-propfile path-to-properties-file] ] [ -validate ] ] [-help ]</td>
</tr>
<tr>
<td><strong>Windows</strong></td>
<td><code>wsaconfig</code> [ [ [-name wSA-instance-name] ] [ [-propfile path-to-properties-file] ] [ -validate ] ] [-help ]</td>
</tr>
</tbody>
</table>

- **-name wSA-instance-name**

  Specifies which existing WSA configuration to examine. The name must match the name of an existing WSA configuration in the specified properties file. If you do not specify a WSA, the **WSACONFIG** utility analyzes all WSA 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, such as:

  - `OpenEdge-Install-Directory\properties\ubroker.properties` in Windows.
  - `OpenEdge-Install-Directory/properties/ubroker.properties` on UNIX.

- **-validate**

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

- **-help**

  Displays command-line help.

**Example**

The following command validates the syntax and views the configurations of all WSA instances defined within the test.properties file located in the current working directory:

```
wsaconfig -propfile test.properties -validate
```
WSAMAN deploy

Deploys a Web service to a WSA instance.

This transfers the Web service’s WSM file to the WSA instance, which generates the Web Services Application Descriptor (WSAD), WSDL, and friendlyname.props files. Most of the properties in the friendlyname.props file are initialized from the values in the default.props file of the WSA instance you are deploying to.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UNIX</strong></td>
<td>wsaman -name wsinstance-name -wsm wsm-location-on-wsaman-machine [ -appname new-appfriendlyname ] [ -namespace new-targetnamespace ] [ -encoding new-encodingstyle ] -deploy</td>
</tr>
<tr>
<td><strong>Windows</strong></td>
<td></td>
</tr>
</tbody>
</table>

- **-name wsinstance-name**

  The name of the WSA instance where you want to deploy the Web service.

- **-wsm wsm-location-on-wsaman-machine**

  The path of the WSM file on the WSAMAN machine.

- **-appname new-appfriendlyname**

  The new friendly name of the Web service.

**Note:** This value replaces the value supplied by the developer to ProxyGen.

- **-namespace new-targetnamespace**

  The new target namespace of the Web service.

**Note:** This value replaces the value supplied by the developer to ProxyGen.
-encoding new-encodingstyle

An integer indicating the new style/use combination of the WSDL file.

Use a value from Table B–8, which lists the WSDL style/use combinations supported.

Table B–8: Setting the SOAP format for deployment using WSAMAN

<table>
<thead>
<tr>
<th>To specify this WSDL style/use combination . . .</th>
<th>Use this value . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPC/Encoded</td>
<td>1</td>
</tr>
<tr>
<td>RPC/Literal</td>
<td>2</td>
</tr>
<tr>
<td>Document/Literal</td>
<td>3</td>
</tr>
</tbody>
</table>

Notes

- The WSM file must be locally accessible from the machine running OpenEdge Explorer or the WSAMAN utility; that is, located on a local drive or a mapped network drive.

- Values for the appServiceName and appServiceConnectionMode properties are initialized from the WSM file, not from default.props.

- When a Web service is deployed, it is initially disabled.
WSAMAN disable

Makes a deployed Web service temporarily unavailable to incoming client requests.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX Windows</td>
<td><code>wsaman</code> <code>-name wsainstance-name</code> `{ -appname app-friendlname</td>
</tr>
</tbody>
</table>

- **-name wsainstance-name**
  
  The name of the WSA instance to which the Web service is deployed.

- **-appname app-friendlname**
  
  The friendly name of the Web service.

- **-namespace app-targetnamespace**
  
  The target namespace of the Web service.

**Note**

When a Web service is disabled, the WSA begins a shutdown process that involves stopping the Web service’s existing client requests and terminating any AppServer connections in the Web service’s connection pool.
**WSAMAN enable**

Makes a deployed Web service available to incoming client requests.

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UNIX</strong></td>
<td>`wsaman -name wsainstance-name { -appname app-friendlyname</td>
</tr>
<tr>
<td><strong>Windows</strong></td>
<td></td>
</tr>
</tbody>
</table>

- **-name wsainstance-name**
  - The name of a WSA instance.

- **-appname app-friendlyname**
  - The friendly name of the Web service.

- **-namespace app-targetnamespace**
  - The target namespace of the Web service.

**Notes**

- When a Web service is deployed, it is disabled.

- When a Web service is imported, it is disabled.

- While a Web service is enabled, you can set only the `serviceFaultLevel` and `serviceLoggingLevel` properties. To set the other properties, the Web service must be disabled. Any changes to the other properties take effect when the Web service is enabled.

- When a Web service is enabled, the WSA begins a startup process that involves establishing the Web service’s connection pool. Establishing the Web service’s connection pool involves pre-establishing any AppServer connections that will be cached to handle incoming client requests more efficiently.
WSAMAN export

Creates a WSD file on the local system from an existing Web service on an existing WSA instance. This facilitates updating a production Web service, cloning a Web service, or moving a Web service from WSA instance to WSA instance.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX Windows</td>
<td>`wsaman -name wsainstance-name { -appname app-friendlyname</td>
</tr>
</tbody>
</table>

- `name wsainstance-name`
  
  The name of the WSA instance to which the Web service is deployed.

- `appname app-friendlyname`
  
  The friendly name of the Web service.

- `namespace app-targetnamespace`
  
  The target namespace of the Web service.

- `wsd wsd-location-on-wsaman-machine`
  
  The path of the WSD file on the WSAMAN machine.

  If this parameter is not supplied, `-export` creates the file `friendlyname.wsd` in the current working directory.

Note

When a Web service is exported and imported, the imported Web service is disabled and a `friendlyname.props` file identical to the original is created for it.
WSAMAN getdefaults

Displays the default Web service properties associated with a WSA instance. These properties reside in the WSA instance's `default.props` file and are used to initialize the `friendlyname.props` file of each Web service deployed to this WSA instance.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX Windows</td>
<td>wsaman -name wsinstance-name -getdefaults</td>
</tr>
</tbody>
</table>

- `-name wsinstance-name` is the name of a WSA instance.

For more information on these properties, see Appendix A, “Reference to OpenEdge Web Service Properties.”
**WSAMAN getprops (Service)**

Displays the properties of a Web service. These are stored in the Web service’s `friendlyname.props` file.

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UNIX</strong></td>
<td><code>wsaman -name wsainstance-name</code></td>
</tr>
<tr>
<td></td>
<td><code>{    -appname app-friendyname</code></td>
</tr>
<tr>
<td></td>
<td><code>    -namespace app-targetnamespace</code></td>
</tr>
<tr>
<td></td>
<td><code>-getprops</code></td>
</tr>
</tbody>
</table>

- **-name wsainstance-name**
  
The name of the WSA instance to which the Web service is deployed.

- **-appname app-friendyname**
  
The friendly name of the Web service.

- **-namespace app-targetnamespace**
  
The target namespace of the Web service.

For more information on these properties, see Appendix A, “Reference to OpenEdge Web Service Properties.”

**Note**

If the Web service is disabled, all relevant properties are displayed. If the Web service is enabled, only the `serviceFaultLevel` and `serviceLoggingLevel` properties are displayed.
WSAMAN getprops (WSA)

Displays the current value of the properties that can be changed while the WSA is running.

These properties are:

- `loggingLevel`
- `enableWsdl`
- `enableWsdlListings`
- `webAppEnabled`
- `debugClients`
- `logMsgThreshold`

These properties reside in the `ubroker.properties` file. For more information on them, see the comments in `ubroker.properties`.

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UNIX</strong></td>
<td><code>wsaman -name wsinstance-name -getprops</code></td>
</tr>
<tr>
<td><strong>Windows</strong></td>
<td></td>
</tr>
</tbody>
</table>

`-name wsinstance-name`

The name of a WSA instance.

**Note**

The values of these properties can be temporarily overridden through WSAMAN setprops (WSA). Such changes are not written to `ubroker.properties`. If there are such changes in effect, the values displayed by the WSAMAN getprops (WSA) might not match the values in `ubroker.properties`. 
WSAMAN getstats (Service)

Displays the statistics for a Web service.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| UNIX Windows     | `wsaman
- name wsainstance-name
  { -appname app-friendlyname
    | -namespace app-targetnamespace }
- getstats` |

`-name wsainstance-name`

The name of the WSA instance to which the Web service is deployed.

`-appname app-friendlyname`

The friendly name of the Web service.

`-namespace app-targetnamespace`

The target namespace of the Web service.

For more information on these statistics, see the OpenEdge Explorer online help for Web services.
WSAMAN getstats (WSA)

Displays the statistics for a WSA instance.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| UNIX Windows     | **wsaman**
|                  | -name *wsainstance-name* |
|                  | -getstats |

- **name wsainstance-name**

The name of a WSA instance.

For more information on these statistics, see the OpenEdge Explorer online help.
**WSAMAN import**

Deploys a previously exported Web service. This facilitates updating a production Web service, cloning a Web service, or moving a Web service from WSA instance to WSA instance.

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UNIX</strong></td>
<td><code>wsaman</code>&lt;br&gt; <code>-name wsainstance-name</code>&lt;br&gt; <code>-wsd wsd-location-on-wsaman-machine</code>&lt;br&gt; <code>[ -appname new-appfriendlyname ]</code>&lt;br&gt; <code>[ -namespace new-targetnamespace ]</code>&lt;br&gt; <code>[ -encoding new-encoding-style ]</code>&lt;br&gt; <code>-import</code></td>
</tr>
<tr>
<td><strong>Windows</strong></td>
<td><code>wsaman</code>&lt;br&gt; <code>-name wsainstance-name</code>&lt;br&gt; <code>-wsd wsd-location-on-wsaman-machine</code>&lt;br&gt; <code>[ -appname new-appfriendlyname ]</code>&lt;br&gt; <code>[ -namespace new-targetnamespace ]</code>&lt;br&gt; <code>[ -encoding new-encoding-style ]</code>&lt;br&gt; <code>-import</code></td>
</tr>
</tbody>
</table>

- **-name wsainstance-name**
  
The name of the WSA instance where you want to deploy the Web service.

- **-wsd wsd-location-on-wsaman-machine**
  
The path of the WSD file on the WSAMAN machine.

- **-appname new-appfriendlyname**
  
The new friendly name of the Web service.

**Note:** This value replaces the value supplied by the developer to ProxyGen.

- **-namespace new-targetnamespace**
  
The new target namespace of the Web service.

**Note:** This value replaces the value supplied by the developer to ProxyGen.

- **-encoding new-encoding-style**
  
An integer indicating the style/use combination of the WSDL file.

Use a value from **Table B–9**, which lists the WSDL style/use combinations supported.

**Table B–9:** Setting the SOAP format for import using WSAMAN

<table>
<thead>
<tr>
<th>To specify this WSDL style/use combination . . .</th>
<th>Use this value . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPC/Encoded</td>
<td>1</td>
</tr>
<tr>
<td>RPC/Literal</td>
<td>2</td>
</tr>
<tr>
<td>Document/Literal</td>
<td>3</td>
</tr>
</tbody>
</table>
Notes

- The WSD file must be locally accessible from the machine running OpenEdge Explorer and the WSAMAN utility, that is, located on a local drive or a mapped network drive.
- When a Web service is imported, it is disabled.
WSAMAN list

Displays the list of Web service applications that have been deployed to the WSA instance.

For each Web service, the following are displayed:

- Friendly name
- Namespace URI
- Status

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td><code>wsaman -name wsainstance-name -list</code></td>
</tr>
<tr>
<td>Windows</td>
<td><code>wsaman -name wsainstance-name -list</code></td>
</tr>
</tbody>
</table>

`-name wsainstance-name`

The name of a WSA instance.
WSAMAN query (Service)

Displays the following information about a Web service:

- Target NameSpace
- Status (running, not running, or status not known)
- AppServer URL
- Session model
- WSDL style/use
- Web service properties

**Note:** All properties are displayed, regardless of whether the Web service is enabled or disabled.

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>wsaman -name wsainstance-name { -appname app-friendlyname } -query</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
</tbody>
</table>

- **-name wsainstance-name**
  
  The name of the WSA instance to which the Web service is deployed.

- **-appname app-friendlyname**
  
  The friendly name of the Web service.

- **-namespace app-targetnamespace**
  
  The target namespace of the Web service.
WSAMAN query (WSA)

Displays the following characteristics and properties of a WSA instance:

- Status (running, not running, or status not known)
- Whether the Administration function is enabled or disabled
- Whether the WSDL function is enabled or disabled
- Whether the Web service function is enabled or disabled

The properties are stored in the WSA instance’s ubroker.properties file. For more information on them, see the comments in ubroker.properties.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX Windows</td>
<td>wsaman -name wsainstance-name -query</td>
</tr>
</tbody>
</table>

-name wsainstance-name

The name of a WSA instance.
**WSAMAN resetdefaults**

Resets, to its original values, a WSA instance’s `default.props` file (which contains the default Web service properties associated with the WSA instance).

The `default.props` file is used to initialize the `friendlyname.props` file of each Web service deployed to the WSA instance.

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UNIX</strong></td>
<td><code>wsaman -name wsainstance-name -resetdefaults</code></td>
</tr>
<tr>
<td><strong>Windows</strong></td>
<td></td>
</tr>
</tbody>
</table>

`-name wsainstance-name`

The name of a WSA instance.

For more information on these properties, see Appendix A, “Reference to OpenEdge Web Service Properties.”
**WSAMAN resetprops**

Reinitializes a Web service’s *friendlyname*.props file to the current value of the WSA instance’s `default.props` file (which contains the default Web service properties associated with the WSA instance).

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| UNIX Windows     | `wsaman`<br>`-name wsainstance-name`<br>`{-appname app-filename}
|                  |       |
|                  |       |
|                  | `-namespace app-targetnamespace`<br>`-resetprops` |

- `-name wsainstance-name`

  The name of the WSA instance to which the Web service is deployed.

- `-appname app-filename`

  The friendly name of the Web service.

- `-namespace app-targetnamespace`

  The target namespace of the Web service.

For more information on these properties, see Appendix A, “Reference to OpenEdge Web Service Properties.”

**Note**

If the Web service is enabled, the `resetprops` function resets only `serviceFaultLevel` and `serviceLoggingLevel`, which are the only properties of an enabled Web service that can be set.
**WSAMAN resetstats (Service)**

Resets the statistics for a Web service.

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| **UNIX**         | `wsaman -name wsainstance-name
                    { -appname app-friendlname
                       | -namespace app-targetnamespace } -resetstats` |
| **Windows**      | `wsaman -name wsainstance-name
                    { -appname app-friendlname
                       | -namespace app-targetnamespace } -resetstats` |

- `-name wsainstance-name`
  
  The name of the WSA instance to which the Web service is deployed.

- `-appname app-friendlname`
  
  The friendly name of the Web service.

- `-namespace app-targetnamespace`
  
  The target namespace of the Web service.

For more information on these statistics, see the OpenEdge Explorer online help for Web services.
WSAMAN resetstats (WSA)

Resets the statistics for a WSA instance.

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td><code>wsaman -name wsainstance-name -resetstats</code></td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
</tbody>
</table>

- `-name wsainstance-name`  
  The name of a WSA instance.

For more information on these statistics, see the OpenEdge Explorer online help for Web services.
WSAMAN setdefault

Sets one of the default Web service properties associated with a WSA instance.

These properties reside in the default.props file associated with this WSA instance and are used to initialize the friendlyname.props file of each Web service deployed to it.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX Windows</td>
<td>wsaman -name wsainstance-name -prop property-name -value property-value -setdefaults</td>
</tr>
</tbody>
</table>

- name wsainstance-name

The name of a WSA instance to which the Web service is deployed.

-prop property-name

The name of a property.

Note: This parameter is case sensitive.

-value property-value

The value of the property.

For more information on these properties, see Appendix A, “Reference to OpenEdge Web Service Properties.”
WSAMAN setprops (Service)

Sets one of the properties of a Web service. These are stored in the Web service’s friendlyname.props file.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UNIX</strong></td>
<td>-name wsainstance-name</td>
</tr>
<tr>
<td></td>
<td>{ -appname app-friendlyname</td>
</tr>
<tr>
<td></td>
<td>-prop property-name</td>
</tr>
<tr>
<td></td>
<td>-value property-value</td>
</tr>
<tr>
<td></td>
<td>-setprops</td>
</tr>
<tr>
<td><strong>Windows</strong></td>
<td>wsaman -name wsainstance-name</td>
</tr>
<tr>
<td></td>
<td>{ -appname app-friendlyname</td>
</tr>
<tr>
<td></td>
<td>-prop property-name</td>
</tr>
<tr>
<td></td>
<td>-value property-value</td>
</tr>
<tr>
<td></td>
<td>-setprops</td>
</tr>
</tbody>
</table>

-name wsainstance-name

The name of the WSA instance to which the Web service is deployed.

-appname app-friendlyname

The friendly name of the Web service.

-namespace app-targetnamespace

The target namespace of the Web service.

-prop property-name

The name of a property.

Note: This parameter is case sensitive.

-value property-value

The value of a property.

For more information on these properties, see Appendix A, “Reference to OpenEdge Web Service Properties.”

Note: To set any Web service property except for serviceFaultLevel and serviceLoggingLevel, the Web service must be disabled. Otherwise, the request fails.
**WSAMAN setprops (WSA)**

Temporarily changes the value of one of the properties of a WSA instance.

These properties are:

- `loggingLevel`
- `enableWsdl`
- `enableWsdlListings`
- `webAppEnabled`
- `debugClients`
- `logMsgThreshold`

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td><code>wsaman</code>&lt;br&gt;<code>-name wsainstance-name</code>&lt;br&gt;<code>-prop property-name</code>&lt;br&gt;<code>-value property-value</code>&lt;br&gt;<code>-setprops</code></td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
</tbody>
</table>

- `name wsainstance-name`
  The name of a WSA instance.

- `prop property-name`
  The name of a property.

  **Note:** This parameter is case sensitive.

- `value property-value`
  The value of the property.

**Notes**

- Changes for the `setprops` function take effect immediately.
- Although the properties set by `setprops` reside in `ubroker.properties`, `setprops` changes are not written to `ubroker.properties`.
- Once a `setprops` change takes effect, it lasts until the JSE is restarted.
- For more information on the properties, see the comments in `ubroker.properties`. 
WSAMAN undeploy

Undeploys a Web service from a WSA instance. This removes the Web service’s WSDL, WSAD, and friendlyname.props files.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX Windows</td>
<td>wsaman -name wsainstance-name {-appname app-friendlyname</td>
</tr>
</tbody>
</table>

-name wsainstance-name

The name of the WSA instance to which the Web service is deployed.

-appname app-friendlyname

The friendly name of the Web service.

-namespace app-targetnamespace

The target namespace of the Web service.

Note

When you undeploy a Web service, whether you use the WSAMAN utility or OpenEdge Explorer, no WSD file is created and no information on the undeployed Web service is saved. To create a WSD file, export the Web service before undeploying it.
WSAMAN update

Lets you change a Web service’s deployment information (stored in the WSM file) without undeploying and deploying. Updating does not affect the Web service’s friendlyname.props file. It is useful during development, but not during production.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| UNIX Windows     | `wsaman`  
|                  | `-name `wsainstance-name`  
|                  | `-wsm wsm-location-on-wsaman-machine`  
|                  | `{ `-appname app-friendlyname`  
|                  | | `-namespace app-targetnamespace` }  
|                  | `[ `-encoding encodingstyle `]  
|                  | `-update` |

`-name wsainstance-name`

The name of the WSA instance to which the Web service is deployed.

`-wsm wsm-location-on-wsaman-machine`

The path of the WSM file on the WSAMAN machine.

`-appname app-friendlyname`

The friendly name of the Web service.

`-namespace app-targetnamespace`

The target namespace of the Web service.

`-encoding encodingstyle`

An integer indicating the new style/use combination of the WSDL file.

Use a value from Table B–10, which lists the WSDL style/use combinations supported.

<table>
<thead>
<tr>
<th>To specify this WSDL style/use combination . . .</th>
<th>Use this value . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPC/Encoded</td>
<td>1</td>
</tr>
<tr>
<td>RPC/Literal</td>
<td>2</td>
</tr>
<tr>
<td>Document/Literal</td>
<td>3</td>
</tr>
</tbody>
</table>

Notes

- If the friendly name is different from the AppObject name, you must use the `-appname` option and not the `-namespace` option.

- Progress Software Corporation recommends that you never use the WSAMAN update function on a production system.
WSCONFIG

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. (WebSpeed only).

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UNIX</strong></td>
<td>wsconfig [ [ [ -name component-name ] [ -propfile path-to-properties-file ] [ -messenger ] [ -validate ] ] [ -help ] ]</td>
</tr>
<tr>
<td><strong>Windows</strong></td>
<td></td>
</tr>
</tbody>
</table>

- **-name component-name**

  Specifies the name of an existing WebSpeed Transaction Server or Messenger configuration to examine. The name must match a 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**

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

- **-help**

  Displays command-line help.
**Examples**  
Table B–11 shows several examples that use the WCONFIG command. Assume the Transaction Server name is `wsbroker1`.

**Table B–11: WCONFIG 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 defined within a different property file.</td>
<td><code>wsconfig -propfile g:\other.properties -validate</code></td>
</tr>
</tbody>
</table>
Controls 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 (WebSpeed only).

### Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX Windows</td>
<td><code>wtbman {</code></td>
</tr>
<tr>
<td></td>
<td><code>{ -name transaction-server-name</code></td>
</tr>
<tr>
<td></td>
<td>`{</td>
</tr>
<tr>
<td></td>
<td>`</td>
</tr>
<tr>
<td></td>
<td>`</td>
</tr>
<tr>
<td></td>
<td>`</td>
</tr>
<tr>
<td></td>
<td>`[ -host host-name -user user-name</td>
</tr>
<tr>
<td></td>
<td><code>[ -port port-number ]</code></td>
</tr>
<tr>
<td></td>
<td>`}</td>
</tr>
</tbody>
</table>

- **-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.

- **-listallprops**
  
  Displays all active broker properties, including updated values for dynamic properties that have changed.

- **-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 in to 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.

-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

Table B–12 shows 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 B–12:  WTBMAN command examples

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

1. Prompts for a password.
Note

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

- 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.

- 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.

- 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.
Dynamic properties are server properties (ubroker properties) that can be changed after an AppServer instance has been started.

For DataServers, agents do not use property files to start up, but the brokers that start them do. Dynamic properties that affect DataServers are modified the broker and any new agents will use the new values.

This reference lists all the dynamic properties, what area of the server they affect, and how changes to that property affect the server.

**Note:** Some of the properties used to support Progress Actional support for OpenEdge monitoring are also dynamic. You can find this information in the “Enabling Actional monitoring of AppServer resources” section on page 2–42

This appendix contains the following sections:

- Overview
- Dynamic properties
Overview

Properties are generally set by an administrator before an AppServer broker is started and those property values are used as long as the broker runs. To change these property values, an AppServer broker must be stopped first, have its properties updated, and then be restarted.

Some properties, called *dynamic properties*, can be changed while the AppServer broker is running. How a change to one of these properties affects the server depends on the particular property. Some properties when changed will be available immediately for reference by running agents as well as newly launched agents. Other properties, if changed, will only affect newly launched agents. Existing agents will continue to use the value they obtained when they were started.

DataServer brokers also start by reading server properties from the ubroker.properties file. Dynamic properties that affect DataServer brokers pass on any changed values to newly started DataServer agents.

Each AppServer has a master property that enables or disables the use of dynamic property updates for that server. The property called `allowRuntimeUpdates` controls whether an AppServer allows runtime updates to properties. It is located in the [UBroker] section of the ubroker.properties file. *Table C–1* describes the valid settings for this property:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Runtime property changes will not be allowed for this application server. This is the default value.</td>
</tr>
<tr>
<td>1</td>
<td>Runtime property changes will be allowed for this application server. The AppServer will listen for changes to the ubroker.properties file and reload all value and behavioral properties. This is true whether the file is directly edited or changed through Progress Explorer, OpenEdge Explorer, or any other tool.</td>
</tr>
</tbody>
</table>
Dynamic properties

This section lists properties that are dynamic. The subsections arrange the properties in useful groupings that describe the effect of changing the listed properties at runtime.

The following parts of the AppServer environment support some dynamic properties:

- NameServer
- Universal Broker
- AppServer agent
- WebSpeed agent
- SonicMQ Broker Adapter

**General dynamic properties: AppServer and WebSpeed**

Any changes to the following properties affect all current and new agents:

- AutoTrimTimeout
- collectStatsData
- flushStatsData
- connectingTimeout
- requestTimeout

**General dynamic properties: Appserver, WebSpeed, and DataServer**

Any changes to the following properties only affect new agents that are started after these values have been changed because they affect how the agent is started. Existing (running) brokers and agents are not updated. Also note that DataServer agents do not have configuration property files. DataServer agents pick up changed values from new DataServer brokers that do read the properties file.

- srvrExecFile
- srvrMaxPort
- srvrMinPort
- srvrStartupScriptParam
- srvrStartupTimeout

**General dynamic properties: Appserver agent**

Any changes to the following properties affect all current and new agents:

- srvrActivateProc
- srvrConnectProc
- srvrDeactivateProc
- srvrDisconnProc
- srvrShutdownProc
Any changes to the following properties only affect new agents that are started after these values have been changed because they affect how the agent is started. Existing (running) agents are not updated:

- `srvrStartupProc`
- `srvrStartupProcParam`

**Logging dynamic properties: Appserver and WebSpeed**

The following properties affect log file information. Any changes will affect all current and new brokers and agents:

- `brkrLoggingLevel`
- `brkrLogEntryTypes`
- `srvrLoggingLevel`
- `srvrLogEntryTypes`

**General dynamic property: AppServer broker and registered NameServer**

Any changes affects all current and new brokers as well as the broker's registered NameServer:

- `priorityWeight`

**General dynamic property: SonicMQ Adapter**

Any changes to the following property affects all current and new broker adapters:

- `srvrLoggingLevel`

**General dynamic properties: NameServer**

Any changes to the following properties affect all current and new NameServers:

- `loggingLevel`
- `logEntryTypes`
- `brokerKeepAliveTimeout`
Index

A

Acceptor, HTTP/SOAP
adding to SonicMQ broker 16–16

Actional 2–42

Actional interceptors
AppServer Internet adapter 2–45
OpenEdge Adapter for Sonic 2–43
OpenEdge Adapter for Sonic MQ 2–43
OpenEdge AppServer 2–42
OpenEdge batch client 2–45
Web services adapter 2–44
Web services out 2–46
WebSpeed 2–44

Active Server Page Messenger 14–2

adaptconfig tool
for BrokerConnect configuration 15–9

ADAPTCONFIG utility B–3
note B–3
syntax B–3

adaptman tool
for BrokerConnect administration 15–9

ADAPTMAN utility B–4
example B–5
note B–5
syntax B–4

Administration framework
AppServer 2–10
preparing to use 2–15
WebSpeed in Windows 9–4
WebSpeed on UNIX 10–4

Administrative tasks
configuring AppServer components 2–16, 3–7
setting up the execution environment 2–27
starting and managing AppServers 2–29
UNIX requirements 2–16, 3–7
using command-line utilities 2–31
using Progress Explorer 2–29
Windows requirements 2–16
summary 2–47

AdminServer
defined
WebSpeed in Windows 9–4
WebSpeed on UNIX 10–4
definition
AppServer 2–10
PROADSV utility B–19
starting in Windows 2–48, 9–20
starting on UNIX 2–48, 10–16, B–19
starting with PROADSV utility 10–24

Advanced Features
WebSpeed broker setting 9–7, 9–10
WebSpeed setting 9–8, 9–11

Agent
AVAILABLE status
WebSpeed in Windows 9–33
WebSpeed on UNIX 10–26
BUSY status
WebSpeed in Windows 9–33
WebSpeed on UNIX 10–26
code page 12–11
error messages 12–11
international configurations 12–9
language 12–11
LIMBO status
WebSpeed in Windows 9–33
WebSpeed on UNIX 10–26

LOCKED status
WebSpeed in Windows 9–33
WebSpeed on UNIX 10–26
starting dynamically
WebSpeed in Windows 9–34
WebSpeed on UNIX 10–27

STARTING status
WebSpeed in Windows 9–33
WebSpeed on UNIX 10–26
trimming running
WebSpeed in Windows 9–34
WebSpeed on UNIX 10–27

Agent application mode 13–2

AIA
Advanced features 3–9, 3–11
AIACONFIG Utility B–6
collection timeout 3–8, 3–10
SSL 3–9, 3–10

AIA. See AppServer Internet Adapter (AIA)

AIACONFIG utility B–6
note B–6
syntax B–6

Application Internet Adapter
AIACONFIG utility B–6

Application PROPATH
AppServer setting 2–19, 2–22, 9–7, 9–10

Application Service
assigning names
using Progress Explorer 2–18, 2–21,
9–6, 9–10
overview 2–4

Application Service names
AppServer setting 2–18, 2–21, 9–6, 9–10

AppServer 2–1
administration framework 2–10
ASBMAN utility B–7
ASCONFIG utility B–12
code-page management 2–34
component distribution 2–2
configuring
using Progress Explorer 2–20, 3–8,
3–9
configuring the components 2–16, 3–7
fault-tolerant instances 2–6
fault-tolerant NameServers 2–5
load balancing, overview 2–6
operating modes
overview 2–7
overview 1–2
registration overview 2–9
run-time components 2–2
session startup parameters 2–9
starting and managing 2–29
startup and shutdown, overview 2–9
status checking 2–32, 2–49

AppServer agent
administration 2–14
customizing the executable 2–28
definition 2–3

AppServer broker
definition 2–4

AppServer execution
preparing the environment 2–27

AppServer Internet Adapter (AIA)
configuration information 3–14
configuring 3–7
collection status 3–13
general administration 3–2
installation 3–2
instance name 3–4
overview 1–3
testing configuration 3–5, 3–13
testing utilities 3–13

AppServiceHost service property A–5
appServiceName service property A–5
appServicePort service property A–5

Audio files (.au)
stored on Web server
UNIX 10–12
Windows 9–18

Authentication
WSA 4–5

Authorization
WSA 4–5

Auto startup
AppServer setting 2–18, 2–21, 9–6, 9–9

AVAILABLE status 2–32
B

BrokerConnect 3–3
  adaptconfig tool B–3
  ADAPTCONFIG Utility B–3
  ADAPTCONFIG utility B–3
  adaptman tool 15–9, B–4
  ADAPTMAN Utility B–4
  ADAPTMAN utility B–4
  and NameServer 15–6
  attributes specified in ubroker.properties
  15–7
  Command line configuration 15–9
  Configuring 15–5
  connection timeout 3–8, 3–10
  HTTP and HTTPS tunneling 15–3
  security 15–3
  SSL 3–9, 3–10, 15–3
  ubroker.properties 15–7

BUSY status 2–32

C

CGI Messenger
  configuring on UNIX 10–9
  executable
  UNIX 10–6
  Windows 9–14
  invoking on UNIX 10–10

Client application
  definition 2–3

Client applications
  administration 2–11

Client port range 13–19

ClientConnect
  Configuring 15–4
  SMQConnect 15–4

Code page
  used by JMS client 15–13

Code pages 12–11
  client and AppServer 2–34
  run-time character conversions 2–35
  settings 2–34

Command line tools
  adaptconfig B–3
  adaptman 15–9, B–4

Command-line utilities
  NSCONFIG
    UNIX 10–5
    Windows 9–13
  NSMAN

  UNIX 10–5
  Windows 9–13
  summary 2–48
  WSCONFIG
    UNIX 10–5
    Windows 9–13
  WTBMAN
    UNIX 10–5
    Windows 9–12

Component distribution 2–5

Configuration information
  for AIA 3–14

Configurations
  Messenger-only installs
    UNIX 10–23
    Windows 9–30
  UNIX overview 10–2
  validating with status.p
    UNIX 10–26
    Windows 9–33
    Windows overview 9–2

Configuring
  AIA 3–7
  AppServers
    in ubroker.properties file 2–14, 2–24,
    3–11
    using Progress Explorer 2–20, 3–8,
    3–9
  NameServers
    using Progress Explorer 2–17, 3–8
  WebSpeed
    using Progress Explorer 9–6, 9–9
  WSA for Web services 5–15
  WSA instances 5–12

CONNECT statement 12–2

Connection status
  for AIA 3–13

Connection-level fault tolerance, definition
  2–4

connectionLifetime service property A–6

Controlling NameServer
  AppServer setting 2–18, 2–21, 9–6, 9–9
  definition 2–4

Creating
  Session objects 15–4
  WSA instances 5–8
  multiple 5–9

Customizing the AppServer agent
  executable 2–28
Index

D

Database
  code page 12–11
  connecting locally 12–3
  connecting programatically 12–2
  connecting remotely 12–3
  multi-national 12–8
  shared memory connections 12–3
  Sports2000 12–2

Database server
  starting 12–2

Data Servers
  and WebSpeed architecture 12–6
  connecting with 12–5
  local 12–6
  managing agent connections 12–7
  remote 12–6

Date (-d) startup parameter 12–11

DB2 database
  connecting to 12–5

DB2/400 database
  connecting to 12–5

Debug mode 13–6

Debugger enabling
  AppServer setting 2–19, 2–23
  WebSpeed setting 9–8, 9–11

Default service
  AppServer setting 2–18, 2–21
  WebSpeed setting 9–6, 9–10

deploy function
  WSAMAN utility B–21

Deploying Web services 6–4

Deployment directory
  changing for Web services 6–6

Deployment files
  OpenEdge Web service 6–5
  Web service management 6–10

Deployment information
  for Sonic ESB services 16–7, 16–9

Development mode
  WebSpeed security 13–2

Digital certificates
  and Web servers 3–6
  installing for WSA 4–6

Directory
  changing for Web service deployment 6–6

disable function
  WSAMAN utility B–23

DLC environment variable
  setting for NSAPI Web server
    UNIX 10–8

Document root directory
  static HTML files in
    UNIX 10–12
    Windows 9–18

Dynamic code-page support 11–1

E

enable function
  WSAMAN utility B–24

Enabling
  OpenEdge Web services 6–8

Endpoints
  SonicMQ 16–10

Environment variables
  BrokerConnect setting 15–7
    for JSE 3–4
  NameServer setting 2–20, 2–24, 9–8,
    9–12
  PROMSGS 12–9
  PROPATH
    Windows 9–21
    where to set 2–27
  WRKDIR
    UNIX 10–17
    Windows 9–21

ESBOEGEN utility B–13

European Numeric Format (-E) startup
  parameter 12–11

export function
  WSAMAN utility B–25

Exporting
  Web services
  Importing
    Web services 6–10

F

Fault-tolerant AppServers
  overview 2–6

Fault-tolerant NameServers
  overview 2–5

file upload directory 13–9
Files
   log 2–27
   ubroker.properties 2–14
Firewalls 13–15

G
GENUUID utility
   UNIX 10–14
getdefaults function
   WSAMAN utility B–26
getprops (Service) function
   WSAMAN utility B–27
getprops (WSA) function
   WSAMAN utility B–28
getstats (Service) function
   WSAMAN utility B–29
getstats (WSA) function
   WSAMAN utility B–30

H
Host Name (-H) startup parameter
   database connections 12–3
   remote database connections 12–3
HTTP/SOAP acceptor
   adding to SonicMQ broker 16–16
HTTPS 13–12
HTTPS protocol
   enabling for WSA 4–6
   for WSA 4–6

I
idleSessionTimeout service property A–6
Image files (.gif)
   stored on Web server
      UNIX 10–12
      Windows 9–18
import function
   WSAMAN utility B–31
Informix database
   connecting to 12–5
Initial servers to start 2–33
Initialization File (-ininame) startup parameter 12–11
initialSessions service property A–6
Installation
   OpenEdge Adapter for Sonic ESB 16–3
   OpenEdge Adapter for SonicMQ 15–4
Installing and configuring a JSE 3–5
Instance name
   for AIA 3–4
Internal Code Page (-cpinternal) startup parameter 12–11
Internationalization
   code-page management 2–34
      OpenEdge Adapter for SonicMQ 15–13
Internet Protocol List 13–5
ISAPI Messenger
   executable
      Windows 9–14

J
Java
   class files location
      UNIX 10–12
      Windows 9–18
Java Message Service (JMS)
   MapMessage 15–10
   performance considerations 15–10 to 15–11
   reusing messages 15–11
   StreamMessage 15–10
   TextMessage 15–10
Java servlet engine (JSE)
   configurations with WSA 5–3
   configuring to recognize the WSA 4–3
   installing and configuring 3–5
   large Web service messages 4–4
   role in WSA security 7–2
   security files 7–3
   using with uri 3–13
   virtual paths 3–4, 3–5
Java virtual machine (JVM)
   memory management 4–4
JMS messaging 15–10
   calls 15–10
   discardable messages 15–11
   internationalization 15–13
   load balancing 15–11
   message reuse 15–11
   performance 15–10
JSE. See Java servlet engine (JSE)
JVM. See Java virtual machine (JVM)
L

LIMBO status 2–32
list function
  WSAMAN utility B–33

Load balancing
  NameServer
    UNIX 10–19
    Windows 9–25
  overview 2–6

LOCKED status 2–32

Log files 2–27
  configuring
    UNIX 10–17
    Windows 9–21
  maintaining
    UNIX 10–22
    Windows 9–29

Logging properties
  AIA broker setting 3–9, 3–10
  AppServer agent process setting 2–19, 2–22
  AppServer broker setting 2–18, 2–22
  WebSpeed agent process setting 9–7, 9–11
  WebSpeed broker setting 9–7, 9–10

M

Maximum servers kept running 2–33
  minSessions service property A–7

Messaging
  ServerConnect 2–20, 2–23, 9–8, 9–12

Messenger
  downloading executables
    UNIX 10–6
    Windows 9–15
  executable location in Windows 9–14
  executable location on UNIX 10–6
  managing
    UNIX 10–29
    Windows 9–36
  operating system compatibility
    UNIX 10–6
    Windows 9–15
  overview 1–5
  script file location
    UNIX 10–11
    Windows 9–17

Microsoft IIS
  Messenger executable
    Windows 9–14

Minimum servers kept running 2–33

N

NameServer
  administration 2–13
  client port range 13–19
  configuring
    using Progress Explorer 2–17, 3–8
    controlling AIA instance 3–9, 3–10
    controlling, overview 2–4
  fault tolerance 2–5
  load balancing
    UNIX 10–19
    Windows 9–25
  neighbors, overview 2–5
  replication, overview 2–5
  starting
    Windows 9–27
  validating configurations
    UNIX 10–15

NAT 13–16

Native Invocation
  OpenEdge Adapter for Sonic ESB 16–2

Neighbor NameServers
  overview 2–5

Netscape
  Messenger executable
    UNIX 10–6
    Windows 9–14

Network Address Translation 13–16

NSAPI Messenger
  executable
    UNIX 10–6
    Windows 9–14
  settings on UNIX 10–8

NSAPI-type Web server
  configuring WebSpeed on
    UNIX 10–7
    Windows 9–15
  obj.conf file
    UNIX 10–8
  start file
    UNIX 10–8
  nsClientMaxPort service property A–8
nsClientMinPort service property A–8
nsClientPicklistExpiration service property A–8
nsClientPicklistSize service property A–8
nsClientPortRetry service property A–9
nsClientPortRetryInterval service property A–9
NSCONFIG utility 2–15, B–15
  described
    UNIX 10–5
    Windows 9–13
  example B–16
  note B–15
  syntax B–15
NSMAN utility 2–13, B–17
  described
    UNIX 10–5
    Windows 9–13
  example B–18
  note B–18
  syntax B–17
NT requirements 2–16
n-tier deployment
  Messenger-only installs
    UNIX 10–23
    Windows 9–30

O

obj.conf file
  modifying
    UNIX 10–7
    Windows 9–15
  sample
    UNIX 10–8
    Windows 9–16
OpenEdge Adapter for Sonic ESB 16–1 to 16–21
  See Services, SonicESB; Sonic ESB
default service properties 16–8
editing an instances of a service 16–15
editing service properties 16–7, 16–8
exposing service 16–16
installation 16–3
installing 16–3
introducing 16–2
Native Invocation 16–2
overview 1–7
Resource Editor 16–13
security 16–20
service instance 16–10
service instance, creating 16–10
service properties 16–7, 16–15
SSL 16–21
  using 16–7
Web Service Invocation 16–2
  and Web services 16–16
OpenEdge Adapter for SonicMQ 15–1 to 15–13
  BrokerConnect 15–2
  Client connections 15–2
  ClientConnect 15–2
  Code-page encoding 15–13
  command line tools 15–9
  Configuring 15–4
  Configuring BrokerConnect 15–5, 15–7, 15–9
  Configuring ClientConnect 15–4
  Configuring ServerConnect 15–4, 15–5
  environment variables for 15–7
  installation 15–4
  internationalization 15–13
  introducing 15–2
  JMS message size 15–10
  maximizing performance 15–10
  overview 1–6
  reusing messages 15–11
  server startup parameters 15–6, 15–8
  ServerConnect 15–2
OpenEdge RDBMS
  connecting in WebSpeed 12–2
OpenEdge service instance 16–10
OpenEdge startup parameters 2–9
OpenEdge Web service
  deploying 6–4
  properties
    setting A–4
    summary overview A–2
    properties reference A–1
OpenEdge Web services
  administration scenarios 6–12
  changing deployment directory 6–6
default properties 6–2
deployment 6–1
deployment file management 6–10
  enabling 6–8
  Exporting and importing 6–10
  managing post-deployment 6–9
  monitoring and tuning 6–11
  post deployment 6–5
  versioning 6–4
Operating Mode
  AppServer setting 2–18, 2–21, 9–6, 9–9
Operating modes
  overview 2–7
ORACLE database
  connecting to 12–5
Owner Info
AppServer setting 2–18, 2–21
WebSpeed setting 9–6, 9–9

P
Parameter file
starting database with 12–2
passwords 13–10
Performance
JMS messaging 15–10
Port Number
AppServer setting 2–18, 2–21, 9–6, 9–9
port numbers 13–8
Ports
TCP 13–15
Post-deployment
Web service management 6–9
priorityWeight property
UNIX 10–19
Windows 9–25
PROADSV utility B–19
example B–19
starting AdminServer
UNIX 10–4, 10–24
syntax B–19
PROCFG environment variable
setting for NSAPI Web server
UNIX 10–8
Production mode
WebSpeed security 13–2
Progress Actional 2–42
Progress Explorer
BrokerConnect configuration 15–5
configuring 9–5, 10–4
AppServers 2–20, 3–8, 3–9
NameServers 2–17, 3–8
WebSpeed 9–6, 9–9
configuring an AppServer instance 2–16, 3–7
described
Windows 9–5, 10–4
overview 2–12
starting and managing AppServers 2–29
using to edit configuration
UNIX 10–13
Windows 9–19
using with DataServers 12–7
validating configurations
UNIX 10–25
Windows 9–32
PROMGS environment variable
international configurations 12–9
limits on NSAPI Web server 12–9
setting for NSAPI Web server
UNIX 10–8
PROPATH environment variable
overriding 2–19, 2–22, 9–7, 9–10
Windows 9–21
Properties
default Web service 6–2
modifying WSA 5–13
reference for services A–1
Web services deployment file 6–5
Properties file 2–14, 2–24
editing and validating 2–24
guidelines for editing 2–24, 15–7
hierarchy 2–25, 15–7
Property file
overview
UNIX 10–13
Windows 9–19
ptpsession.p 15–4
Q
query (Service) function
WSAMAN utility B–34
query (WSA) function
WSAMAN utility B–35
R
R-code for SmartDataObjects 2–27
Registration mode 13–16
Replicated NameServers, overview 2–5
requestWaitTimeout service property A–9
Requirements. See System requirements
resetdefaults function
WSAMAN utility B–36
resetprops function
WSAMAN utility B–37
resetstats (Service) function
WSAMAN utility B–38
resetstats (WSA) function
WSAMAN utility B–39
Run-time properties
for Sonic ESB services 16–9
S

Sample WSA Web application
moving 4–2

Schema holder 12–6

Script file
wsd_cgi.sh shell 10–9

Secure Socket Layer (SSL)
WSA communications 4–6

Secure Sockets Layer 13–12

Secure Sockets Layer (SSL)
and AIA 3–2
and AppServer 2–20, 2–23, 9–8, 9–11
and OpenEdge Adapter for Sonic ESB 16–20

Security
See Secure Sockets Layer 2–20, 2–23, 9–8, 9–11
JSE initial settings for WSA 7–5
WSA 4–5
WSA configurations 7–1
WSA features 7–3
WSA role 7–2

Server authentication 3–2

Server Control Parameters
AppServer setting 2–19, 2–23, 9–8, 9–11

Server Executable File
AppServer setting 2–19, 2–22
WebSpeed setting 9–7, 9–10

Server Parameters
AppServer setting 2–19, 2–22, 9–7, 9–10

Server Pool Parameters
AppServer setting 2–19, 2–23
choosing the settings 2–33
WebSpeed setting 9–7, 9–11

Server Port Range
AppServer setting 2–19, 2–22, 9–7, 9–10

ServerConnect
Configuring 15–5
configuring 9–5, 10–4, 15–4
Messaging 2–20, 2–23, 9–8, 9–12

Service Name (-S) startup parameter
database connections 12–3
remote database connections 12–3

serviceFaultLevel service property A–10

serviceLoggingLevel service property A–10

Services, Sonic ESB
See OpenEdge Adapter for Sonic ESB;
Web services
deploying 16–18
deployment Information 16–9
instance, creating 16–10
properties, editing 16–7, 16–15
run-time properties 16–9
SOAP Action URI 16–9
SOAP Fault Processing 16–12
Web Service Namespace 16–9

setdefaults function
WSAMAN utility B–40

setprops (Service) function
WSAMAN utility B–41

setprops (WSA) function
WSAMAN utility B–42

Setting CLASSPATH 15–12

Shared memory
database connection 12–3

Single-user database 12–2

SmartDataObject support 2–27

SOAP Action URI
for Sonic ESB service 16–9

SOAP Fault Processing
for Sonic ESB services 16–12

Sonic Management Console
configuration 16–5

Sonic ESB
See OpenEdge Adapter for Sonic ESB
deploying services 16–18
Directory Service 16–8

SonicMQ
See OpenEdge Adapter for SonicMQ;
OpenEdge Adapter for Sonic ESB
Discardable messages 15–11
endpoints 16–10
HTTP/SOAP acceptor, adding 16–16
Load balancing 15–11
Message reuse 15–11

Sports2000 database 12–2

SSL 13–12
BrokerConnect 3–3

SSL advanced feature
AppServer setting 2–20, 2–23
WebSpeed setting 9–8, 9–11

SSL general features
AppServer setting 2–20, 2–23
WebSpeed setting 9–8, 9–11
SSL parameters
  OpenEdge Adapter for SonicMQ setting 15–7

SSL. See Secure Sockets Layer

staleO4GObjectTimeout service property A–10

Starting an AppServer
  NSMAN command-line utility 2–31
  Progress Explorer 2–29

Starting configurations
  WSA
    Starting a WSA instance 5–10

STARTING status 2–32

State-aware operating mode
  overview 2–7

State-free operating mode
  overview 2–7

Stateless operating mode
  overview 2–7

State-reset operating mode
  overview 2–7

Static files
  HTML
    UNIX 10–12
    Windows 9–18
  Java class files
    UNIX 10–12
    Windows 9–18

Status, AppServer run-time 2–32, 2–49

Status.p procedure
  validating configurations
    UNIX 10–26
    Windows 9–33

Stream Code Page (-cpstream) startup parameter 12–11

Syntax
  ASBMAN utility 2–32
  ASCONFIG utility 2–26, 5–14

System requirements
  NT and Windows 2–16
  UNIX 2–16, 3–7

T

TCP ports 13–15

Testing configuration
  for AIA 3–5, 3–13

Testing utilities
  for AIA 3–13

Tuning
  OpenEdge Web services 6–11

U

ubroker.properties file 2–14
  See Properties file
    BrokerConnect properties 15–7
    editing
      UNIX 10–13
      hierarchy
        AppServer 2–25
        UNIX 10–13
      location
        UNIX 10–13
      modifying for BrokerConnect 15–7
      overview
        UNIX 10–13
        Windows 9–19
      structure 10–13
      unique values needed 10–14
      updating for a secure WSA 4–6
      validating edits 10–15

undeploy function
  WSAMAN utility B–43

UNIX requirements 2–16, 3–7

update function
  WSAMAN utility B–44

URL
  international Web site 12–11

URLs
  WSA 5–4

uuid
  in ubroker.properties 10–14

V

Validating
  WebSpeed configurations
    UNIX 10–25
    Windows 9–32

Versioning
  OpenEdge Web services 6–4

Video files (.mpeg, .mpg)
  stored on Web server
    UNIX 10–12
    Windows 9–18

Viewing AIA status information
  using a JSE 3–13
Virtual paths
  for JSE 3–5
  for JSEs 3–4
  using with JSE 3–5
  using with uri 3–5

W

waitIfBusy service property A–11

Web application definition
  WSA 5–6, 5–7

Web server
  configurations with WSA 5–3
  NSAPI type
    UNIX 10–7
    Windows 9–15
  setting up
    UNIX 10–6
    Windows 9–14

Web servers
  and digital certificates 3–6
  server authentication 3–2

Web Service Definition (WSD) file
  and OpenEdge Adapter for Sonic ESB
    16–7

Web Service Description Language file
  deployment to WSA 6–5
  providing to clients 6–10

Web Service Invocation
  OpenEdge Adapter for Sonic ESB 16–2

Web Service Namespace
  for Sonic ESB service 16–9

Web services
  large messages in JSE 4–4
  See OpenEdge Web services
  and OpenEdge Adapter for Sonic ESB
    16–16

Web Services Adapter
  WSACONFIG utility B–20
  WSAMAN utility
    deploy B–21
    disable B–23
    enable B–24
    export B–25
    getdefaults B–26
    getprops (Service) B–27
    getprops (WSA) B–28
    getstats (Service) B–29
    getstats (WSA) B–30
    import B–31
    list B–33
    query (Service) B–34
    query (WSA) B–35
    resetdefaults B–36
    resetprops B–37
    resetstats (Service) B–38
    resetstats (WSA) B–39
    setdefault B–40
    setprops (Service) B–41
    setprops (WSA) B–42
    undeploy B–43
    update B–44

Web Services Adapter (WSA)
  administration
    architecture 5–2
    components 5–3
    prerequisites 5–2
  component description 5–1
  configuration
    levels of 4–1
  configuring for Web services 5–15
  creating instances 5–8
  multiple 5–9
  deploying Web services 6–1
  environments
    production 4–2
  instance
    Properties
      for WSA instance 5–12
    instance management 5–12
  JSE/Web server configurations 5–3
  overview 1–4
  properties
    modifying 5–13
    recognize from JSE 4–3
  role in security 7–2
  Security configurations 7–1
  security configurations
    reference 7–6
  security features 7–3
  security files 7–3
  security initial settings 7–5
  security overview 4–5
  Starting an instance 5–10
  statistics
    WSA 5–14
  testing an instance
    Testing configuration
      for WSA 5–10
  URL construction 5–4
  Web application definition 5–6, 5–7
  Web application location 4–2
  Web service properties, default 6–2
  WSAMAN utility 8–2

Web Services Mapping (WSM) file
  and OpenEdge Adapter for Sonic ESB
    16–7

Web site
  international 12–8
  multi-lingual 12–8
web.xml file
digital certificates 4–6

WebSpeed
agent application mode 13–2
ASP Messenger 14–2
configuration in Windows 9–2
configuration on UNIX 10–2
configuring
using Progress Explorer 9–6, 9–9
Debug mode 13–6
development mode 13–2
firewalls 13–15
Messenger Administration utility 13–4
passwords 13–10
port numbers 13–8
production mode 13–2
registration mode 13–16
running in Windows 9–31
running on UNIX 10–24
security 13–1
testing after configuration
UNIX 10–25
Windows 9–32
WSCONFIG utility B–45
WTBMAN utility B–47

WebSpeed administration framework
UNIX 10–4
Windows 9–4

WebSpeed broker
starting
UNIX 10–20

WebSpeed Messenger. See Messenger

WebSpeed Transaction Server
forcing immediate shut down
UNIX 10–28
Windows 9–35
managing
UNIX 10–27
Windows 9–34
overview 1–5
sample instance
Windows 9–5, 10–4
starting
Windows 9–27
stopping with WTBMAN
UNIX 10–28
Windows 9–35
validating configurations
UNIX 10–15

Weight factor
load balancing
Windows 9–25
setting to zero
Windows 9–26

Weight factors
arbitrary
Windows 9–25
fail-over
Windows 9–26
percentage-based
Windows 9–25

Windows requirements 2–16

Working directory
AppServer setting 2–18, 2–21
environment variable
UNIX 10–17
Windows 9–21
overview 2–27
WebSpeed setting 9–6, 9–9

WRKDIR environment variable
setting for NSAPI Web server
UNIX 10–8

WSA
WSACONFIG utility B–20
WSAMAN utility
deploy B–21
disable B–23
enable B–24
export B–25
getdefaults B–26
getprops (Service) B–27
getprops (WSA) B–28
getstats (Service) B–29
getstats (WSA) B–30
import B–31
list B–33
query (Service) B–34
query (WSA) B–35
resetdefaults B–36
resetprops B–37
resetstats (Service) B–38
resetstats (WSA) B–39
setdefault B–40
setprops (Service) B–41
setprops (WSA) B–42
undeploy B–43
updateWSAMAN utility
update B–44

WSA. See Web Services Adapter (WSA)

WSACONFIG utility B–20
example B–20
syntax B–20
Web Services Adapter (WSA)
WSACONFIG utility 5–14

WSAD file
deployment to WSA 6–5

WSAMAN deploy utility B–21
note B–22
syntax B–21
WSAMAN disable utility B–23
  note B–23
  syntax B–23
WSAMAN enable utility B–24
  syntax B–24
WSAMAN export utility B–25
  syntax B–25
WSAMAN getdefaults utility B–26
  syntax B–26
WSAMAN getprops (Service) utility B–27
  note B–27
  syntax B–27
WSAMAN getprops (WSA) utility B–28
  note B–28
  syntax B–28
WSAMAN getstats (Service) utility B–29
  syntax B–29
WSAMAN getstats (WSA) utility B–30
  syntax B–30
WSAMAN import utility B–31
  note B–32
  syntax B–31
WSAMAN list utility B–33
  syntax B–33
WSAMAN query (Service) utility B–34
  syntax B–34
WSAMAN query (WSA) utility B–35
  syntax B–35
WSAMAN resetdefaults utility B–36
  syntax B–36
WSAMAN resetproprs utility B–37
WSAMAN resetprops utility
  note B–37
  syntax B–37
WSAMAN resetstats (Service) utility B–38
  syntax B–38
WSAMAN resetstats (WSA) utility B–39
  syntax B–39
WSAMAN setdefault utility B–40
  syntax B–40
WSAMAN setprops (Service) utility B–41
  note B–41
  syntax B–41
WSAMAN setprops (WSA) utility B–42
  note B–42
  syntax B–42
WSAMAN undeploy utility B–43
  note B–43
  syntax B–43
WSAMAN update utility B–44
  note B–44
  syntax B–44
WSAMAN utility
  deploy B–21
  note B–22
  syntax B–21
disable B–23
  note B–23
  syntax B–23
enable B–24
  note B–24
  syntax B–24
export B–25
  note B–25
  syntax B–25
getdefaults B–26
  syntax B–26
getprops B–28
getprops (Service) B–27
  note B–27
  syntax B–27
getprops (WSA) B–28
  syntax B–28
getstats (Service) B–29
  syntax B–29
getstats (WSA) B–30
  syntax B–30
import B–31
  note B–32
  syntax B–31
list B–33
  syntax B–33
overview of functions 8–2
query (Service) B–34
  syntax B–34
query (WSA) B–35
  syntax B–35
resetdefaults B–36
  syntax B–36
resetproprs B–37
  note B–37
  syntax B–37
resetstats (Service) B–38
  syntax B–38
resetstats (WSA) B–39
  syntax B–39
setdefault B–40
  syntax B–40
setprops (Service) B–41
  note B–41
  syntax B–41
setprops (WSA) B–42
  note B–42
  syntax B–42
syntax 8–2
Index

undeploy B–43
  note B–43
  syntax B–43, B–44
update B–44
  note B–44

WSASP 14–2

WSASP Messenger
  executable
    Windows 9–14

WCONFIG utility B–45
  described
    UNIX 10–5
    Windows 9–13
  example B–46
  syntax B–45

WSD file. See Web Service Definition file

WSM file. See Web Services Mapping file

WSAdmin 13–4
WBSAdmin page
  URL
    UNIX 10–29
    Windows 9–36
  validating configurations
    UNIX 10–3
    Windows 9–3
  viewing
    UNIX 10–26
    Windows 9–33

WBMAN help 9–35, 10–28
WBMAN utility B–47
  described
    UNIX 10–5
    Windows 9–12
  example B–48
  note B–49
  syntax B–47