OpenEdge® Application Server:
Administration
# Table of Contents

## Preface

Purpose.........................................................................................................................19

Audience.......................................................................................................................20

Organization..................................................................................................................20

References to ABL compiler and run-time features.......................................................22

Using ABL documentation..........................................................................................22

References to ABL data types......................................................................................23

Typographical conventions.........................................................................................23

Examples of syntax descriptions..................................................................................24

Long syntax descriptions split across lines.................................................................26

Complex syntax descriptions with both required and optional elements......................26

Example procedures....................................................................................................26

OpenEdge messages.....................................................................................................27

Obtaining more information about OpenEdge messages..............................................28

## Part I: Introduction

## Chapter 1: Overview of Server and Services Administration

AppServer for OpenEdge...............................................................................................32

AppServer Internet Adapter..........................................................................................32

OpenEdge Web Services Adapter..................................................................................32

WebSpeed Transaction Server and Messenger.............................................................33

OpenEdge Adapter for SonicMQ....................................................................................33

OpenEdge Adapter for Sonic ESB................................................................................34

Configuring the OpenEdge runtime container for Sonic..............................................34

Automatic upgrade of OpenEdge containers .................................................................35

REST Management Agent for REST Web application administration.......................35

## Part II: AppServer and Internet Adapter Administration

## Chapter 2: Configuring and Managing the AppServer

Run-time components and operation..........................................................................40

Distribution of components.........................................................................................42

Fault-tolerant NameServers.........................................................................................43

Fault-tolerant AppServers and load balancing..............................................................43

Operating modes.........................................................................................................44
## Chapter 6: Deploying and Managing OpenEdge Web Services

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initializing a WSA instance's default Web service properties</td>
<td>108</td>
</tr>
<tr>
<td>Security considerations for Web service administration</td>
<td>108</td>
</tr>
<tr>
<td>Deploying a Web service</td>
<td>109</td>
</tr>
<tr>
<td>Versioning Web Services</td>
<td>110</td>
</tr>
<tr>
<td>After deploying Web services</td>
<td>110</td>
</tr>
<tr>
<td>Changing the Web service deployment directory</td>
<td>111</td>
</tr>
<tr>
<td>Enabling a Web service</td>
<td>112</td>
</tr>
<tr>
<td>Administering a deployed Web service</td>
<td>113</td>
</tr>
<tr>
<td>Web service file management for deployed Web services</td>
<td>113</td>
</tr>
<tr>
<td>Options for providing the WSDL file to a client</td>
<td>113</td>
</tr>
<tr>
<td>Exporting and importing Web services</td>
<td>114</td>
</tr>
<tr>
<td>Monitoring and tuning Web services</td>
<td>114</td>
</tr>
<tr>
<td>Typical Web service administration scenarios</td>
<td>115</td>
</tr>
<tr>
<td>Deploying a new version of your Web service</td>
<td>115</td>
</tr>
<tr>
<td>Changing Web service properties</td>
<td>116</td>
</tr>
</tbody>
</table>

## Chapter 7: Web Services Adapter Security Configurations

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>117</td>
</tr>
<tr>
<td>Role of the Java container</td>
<td>117</td>
</tr>
<tr>
<td>Role of the WSA</td>
<td>118</td>
</tr>
<tr>
<td>Java container and WSA security files</td>
<td>118</td>
</tr>
<tr>
<td>Security features available</td>
<td>119</td>
</tr>
<tr>
<td>Initial settings</td>
<td>119</td>
</tr>
<tr>
<td>Alphabetical list of configuration instructions</td>
<td>120</td>
</tr>
<tr>
<td>Controlling access to Web services, WSDL, and WSA administration</td>
<td>121</td>
</tr>
<tr>
<td>Customizing the default administrator's permissions and action settings</td>
<td>124</td>
</tr>
<tr>
<td>Customizing the PSCOper role's permissions and action settings</td>
<td>124</td>
</tr>
<tr>
<td>Customizing WSA administration roles</td>
<td>125</td>
</tr>
<tr>
<td>Disabling access to all Web services, to all WSDL, or to all WSA administration</td>
<td>127</td>
</tr>
<tr>
<td>Disabling authentication for WSA administration</td>
<td>127</td>
</tr>
<tr>
<td>Disabling the deploying and undeploying of Web services</td>
<td>128</td>
</tr>
<tr>
<td>Feature</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Disabling WSDL listings</td>
<td>128</td>
</tr>
<tr>
<td>Enabling authentication for Web services</td>
<td>129</td>
</tr>
<tr>
<td>Enabling authentication for WSDL</td>
<td>130</td>
</tr>
<tr>
<td>Enabling multiple user roles to access Web services, WSDL, or WSA administration</td>
<td>132</td>
</tr>
<tr>
<td>Enabling Web services per user and per application</td>
<td>134</td>
</tr>
</tbody>
</table>

**Chapter 8: Using the WSA Management Utility (WSAMAN).........141**

- Overview of the WSAMAN utility.................................................................141
- Syntax............................................................................................................141
- Functions for managing a WSA instance...................................................142
- Functions for managing a Web service......................................................143

**Part IV: WebSpeed Administration...........................................................145**

**Chapter 9: Configuring WebSpeed in Windows..............................................147**

- WebSpeed configuration overview...............................................................148
- Configuring the WebSpeed components on Windows.....................................148
- WebSpeed administration...............................................................................149
- The AdminService..........................................................................................149
- OpenEdge Management or OpenEdge Explorer..............................................149
- Defining or configuring WebSpeed with OpenEdge Management or OpenEdge Explorer | 150 |
- WebSpeed command-line utilities..................................................................153
- NameServer command-line utilities............................................................153
- Setting up WebSpeed on the Web server machine.......................................154
- Installing the Messenger executable..........................................................154
- Configuring Netscape Web servers for NSAPI Messengers..........................155
- Restarting the Netscape Web server............................................................157
- Placing static files on the Web server.......................................................157
- Overview of the ubroker.properties file.....................................................157
- Starting the AdminService.............................................................................158
- Setting up the WebSpeed environment.........................................................158
- Setting environment variables......................................................................158
- Configuring WebSpeed components.............................................................159
- Eliminating the NameServer..........................................................................160
- Understanding the NameServer's load-balancing option............................161
- Starting the WebSpeed Transaction Server and NameServer........................162
- Maintaining the WebSpeed Transaction Server and NameServer log files........164
- Configuring a WebSpeed Messenger-only installation..................................164
- Starting WebSpeed to test the configuration................................................164
- Testing your configuration............................................................................165
  - Using OpenEdge Management or OpenEdge Explorer to check status...........165
  - Using the query option to check status......................................................165
  - Viewing the Messenger Administration page..........................................166
Running the status.p procedure ....................................................................................166
Managing the WebSpeed Transaction Server ..............................................................166
  Dynamically starting additional agents .................................................................167
  Trimming running agents .....................................................................................167
  AppServer Refresh ...............................................................................................167
  Stopping the WebSpeed broker ..........................................................................167
  Accessing help on WTBMAN ..............................................................................168
Managing the WebSpeed Messenger ........................................................................168

Chapter 10: Configuring WebSpeed on UNIX ............................................................169
  WebSpeed configuration overview ........................................................................170
  Configuring the WebSpeed components on UNIX ..............................................170
  WebSpeed administration .......................................................................................171
    The AdminServer ..............................................................................................171
    OpenEdge Management or OpenEdge Explorer ..............................................171
    WebSpeed command-line utilities ................................................................171
    NameServer command-line utilities .................................................................172
  Setting up WebSpeed on the Web server machine .............................................173
    Installing the Messenger executable ..............................................................173
    Compatibility with the Web server type and operating system ....................173
    Restarting the Netscape Web server ...............................................................176
  Placing static files on the Web server .................................................................178
  Overview of the ubroker.properties file .............................................................179
    The ubroker.properties file structure ..............................................................179
    Editing the ubroker.properties file ................................................................179
  Starting the AdminServer ....................................................................................181
  Setting up the WebSpeed environment ...............................................................181
    Setting environment variables ........................................................................181
    Configuring WebSpeed components ................................................................182
      Load balancing with the NameServer ...........................................................182
  Starting the WebSpeed Transaction Server and NameServer ................................183
  Maintaining the WebSpeed Transaction Server and NameServer log files .........184
  Configuring a WebSpeed Messenger-only installation ......................................184
  Starting WebSpeed to test the configuration .......................................................185
  Testing your configuration ...................................................................................185
    Using OpenEdge Management or OpenEdge Explorer to check status .......185
    Using the query option to check status ...........................................................185
    Viewing the Messenger Administration page ................................................186
    Running the status.p procedure ...................................................................187
  Managing the WebSpeed Transaction Server .....................................................187
    Dynamically starting additional agents ...........................................................187
    Trimming agents ..............................................................................................188
    Stopping the Transaction Server ....................................................................188
    Accessing help on WTBMAN ...........................................................................188
Securing data transmissions between WebSpeed client and server components.............215
Secure HTTP Protocol (HTTPS)..................................................................................215
Secure Sockets Layer (SSL)......................................................................................216
Maximizing WebSpeed compatibility with your firewall.............................................217
WebSpeed TCP port details.......................................................................................217
Important firewall considerations.............................................................................218
Network Address Translation (NAT) and the WebSpeed distributed configuration......218
Using the NameServer client port range value settings with a firewall.....................220

Chapter 14: Using Active Server Pages with WebSpeed.................................223
Setting up the WebSpeed ASP (WSASP) Messenger......................................................223
Accessing the WebSpeed ASP Web page.....................................................................224
Enabling the WebSpeed ASP example.........................................................................225

Part V: Messaging and ESB Administration.....................................................227

Chapter 15: OpenEdge Adapter for SonicMQ Administration..................229
Introducing the OpenEdge Adapter for SonicMQ......................................................229
Client connections to the OpenEdge Adapter for SonicMQ...........................................230
Installing the OpenEdge Adapter for SonicMQ............................................................230
SSL-enabled BrokerConnect operation......................................................................230
HTTPS security for BrokerConnect............................................................................231
Configuring and administering the OpenEdge Adapter for SonicMQ.......................231
Configuring ClientConnect..........................................................................................232
Configuring ServerConnect.......................................................................................232
Configuring BrokerConnect.......................................................................................233
Editing BrokerConnect attributes in ubroker.properties............................................234
Guidelines for editing the properties file.....................................................................235
Configuring BrokerConnect from the command line...............................................236
Maximizing performance.............................................................................................237
Performance comparison.............................................................................................237
Optimizing message size.............................................................................................238
StreamMessage, MapMessage, and TextMessage.......................................................238
Remote and local calls...............................................................................................238
Message reuse............................................................................................................238
Load balancing.............................................................................................................238
Discardable messages.................................................................................................238
Setting the CLASSPATH...............................................................................................239
Internationalization considerations.............................................................................239
Chapter 16: Configuring and Managing the OpenEdge Adapter for Sonic ESB .................................................................241
Introduction to the OpenEdge Adapter for Sonic ESB ..........................................................................................241
Installation of the OpenEdge Adapter for Sonic ESB ..................................................................................242
Configuration .................................................................................................................................................243
Confirming the Sonic Management Console is configured properly (Optional) ..............................................243
Using the OpenEdge Adapter for Sonic ESB .........................................................................................................245
Editing OpenEdge service properties ..............................................................................................................246
Creating an OpenEdge service instance ...........................................................................................................248
Editing an instance of an OpenEdge service ......................................................................................................253
Exposing a service as a standard Web service ..................................................................................................253
Deploying a service instance in Sonic ESB ..........................................................................................................255
Starting the ESB container ................................................................................................................................256
Security considerations for OpenEdge Adapter for Sonic ESB ...........................................................................256
SSL-related service properties ..........................................................................................................................257

Part VI: REST Administration ..........................................................................................................................259

Chapter 17: REST Management Agent Administration .........................................................................................261
Installing and Configuring REST Management Agent .......................................................................................261
Installing a REST Management Agent into a Java container ..............................................................................262
Customizing Apache Tomcat Environment using the protc utility ........................................................................263
Tuning the Java container to handle large Web service messages ....................................................................267
Installing a REST Management Agent on a DMZ server ....................................................................................267
Managing REST Management Agent ..................................................................................................................268
REST Management Agent administration architecture ..........................................................................................269
Starting and testing a REST Management Agent ..............................................................................................273
Managing a REST Management Agent ...............................................................................................................274
Security Configurations for a REST Management Agent ...................................................................................276

Chapter 18: REST Web Application Administration ...............................................................................................277
Managing REST Web applications .......................................................................................................................277
Initializing a REST Web application's runtime properties ..................................................................................278
Deploying a REST Web application ....................................................................................................................278
Enabling a REST Web application ......................................................................................................................279
Generating a REST Web application's WAR and PAAR files ............................................................................279
Republishing a REST Web application ................................................................................................................279
Unpublishing a REST Web service of a REST Web application ........................................................................280
Monitoring and tuning REST Web application ................................................................................................280
Administering a deployed and enabled REST Web application .......................................................................281
Understanding REST Web application URI construction.................................282
Constructing and debugging REST requests to access a REST Web service.........283
SSL support for REST Web applications..............................................................287
SSO support for REST Web applications............................................................288
Debugging HTTP messages of a REST Web application.....................................293
Security configurations for a REST Web application........................................296
Managing REST Web applications in the absence of the REST Management Agent...............................................................296
Deploying the REST Web application...............................................................297
Editing a REST Web application runtime properties........................................297
Editing a REST Web application security configurations................................297
Enabling a REST Web application using Tomcat's manager..............................298
Monitoring and maintaining REST Web application log files........................298
Managing Web server deployed REST Web applications using the REST Management Agent...............................................................298

Chapter 19: Using the RESTMAN Utility.........................................................303
Syntax.............................................................................................................303
Functions for managing a REST Management Agent........................................304
Functions for managing a REST Web application............................................305

Chapter 20: REST Management Agent and REST Web Application

Security Configurations...................................................................................307
Overview........................................................................................................307
Security configuration models.........................................................................308
Working with security configurations.............................................................311
Choosing and applying a security configuration.............................................312
Adding and modifying a user..........................................................................313
Setting hashed and salted password using EncodePassword........................313
Setting hashed and salted passwords using Jacksum.....................................315
Encrypting a password shared by REST Management Agent and REST Web application..................................................................................................................317
Modifying user roles and privileges...............................................................319
Extending CORS support..............................................................................320
Using LDAP in security configurations..........................................................323
Choose and apply an LDAP security configuration........................................324
Configure the LDAP security configuration model.........................................324
Using Single Point of Authentication security configurations..........................324
Configure an OERealm service on an AppServer..........................................326
Select and apply an SPA security configuration.............................................330
Configure the SPA security configuration model...........................................330
Test the OERealm service..............................................................................335
Using SAML in security configurations...........................................................336
Choosing a SAML security configuration model.............................................336
Creating and configuring the Service Provider metadata..............................337
Appendix A: Reference to OpenEdge Web Service Properties............343

Overview..........................................................................................................................343
Summary of properties........................................................................................................344
Setting Service properties................................................................................................347
Alphabetical reference........................................................................................................347
appServerKeepAlive...........................................................................................................348
appServiceConnectionMode...............................................................................................348
appServiceHost..................................................................................................................348
appServiceName.................................................................................................................348
appServicePort..................................................................................................................348
appServiceProtocol...........................................................................................................348
clientASKActivityTimeout.................................................................................................349
clientASKResponseTimeout...............................................................................................349
connectionLifetime...........................................................................................................349
connectionTimeout............................................................................................................349
idleSessionTimeout............................................................................................................350
initialSessions..................................................................................................................350
maxSessions.......................................................................................................................350
minIdleConnections.........................................................................................................350
minSessions.......................................................................................................................351
noHostVerify......................................................................................................................351
noSessionReuse.................................................................................................................351
nsClientMaxPort................................................................................................................351
nsClientMinPort................................................................................................................351
nsClientPicklistExpiration.................................................................................................352
nsClientPicklistSize............................................................................................................352
nsClientPortRetry..............................................................................................................352
nsClientPortRetryInterval.................................................................................................352
requestWaitTimeout.........................................................................................................352
serviceAvailable..............................................................................................................353
serviceFaultLevel.............................................................................................................353
serviceLogEntryTypes.......................................................................................................353
serviceLoggingLevel.........................................................................................................354
sessionMode.....................................................................................................................354
staleO4GLObjectTimeout.................................................................................................354
waitIfBusy........................................................................................................................355

Appendix B: Command and Utility Reference.................................................357
ADAPTCONFIG..................................................................................................................359
ADAPTMAN.........................................................................................................................360
Appendix C: Reference to Dynamic Server Properties.........................421
  Overview..............................................................................................421
  Dynamic properties............................................................................422

Appendix D: OERealm........................................................................425
  Implementing OERealm........................................................................426
    Implement the IHybridRealm Interface..............................................426
    Implementing necessary methods......................................................426
    Implementing security in OE Realm Class(optional).........................427
  Supported OERealm Clients.................................................................427
    Configuring OERealm Clients............................................................428
  OERealm Best Practices.....................................................................430
  Samples.............................................................................................430

Index.................................................................................................431
Preface

For details, see the following topics:

- Purpose
- Audience
- Organization
- References to ABL compiler and run-time features
- Typographical conventions
- Examples of syntax descriptions
- Example procedures
- OpenEdge messages

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®

OpenEdge REST Management Agent for REST Web application administration

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 1, Introduction

Overview of Server and Services Administration on page 31

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 2, AppServer and Internet Adapter Administration

Configuring and Managing the AppServer on page 39

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

Configuring and Managing the AppServer Internet Adapter on page 75

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

Part 3, Web Services Adapter Administration

Configuring a Web Services Adapter Installation on page 89

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

Managing the Web Services Adapter on page 93

Describes how to create and manage a WSA Web application in the context of a Java container, 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.
Deploying and Managing OpenEdge Web Services on page 107
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.

Web Services Adapter Security Configurations on page 117
Describes the components of WSA security and how to create and manage common WSA security configurations, listed for reference by alphabetical order.

Using the WSA Management Utility (WSAMAN) on page 141
Introduces the WSAMAN command-line utility to perform many of the WSA and Web service management tasks that are otherwise performed using OpenEdge Management or OpenEdge Explorer.

Part 4, WebSpeed Administration
Configuring WebSpeed in Windows on page 147
Describes the basic requirements and procedures for configuring a WebSpeed installation in Windows.

Configuring WebSpeed on UNIX on page 169
Describes the basic requirements and procedures for configuring a WebSpeed installation on UNIX.

WebSpeed Dynamic Code-page Support on page 191
Describes how to manage dynamic code-page support for internationalizing WebSpeed installations.

Connecting WebSpeed to a Data Source on page 197
Describes how to connect a WebSpeed application to an OpenEdge data source, such an OpenEdge RDBMS or DataServer.

WebSpeed Security on page 207
Describes the components of WebSpeed security and how to use them to manage secure WebSpeed installations and applications.

Using Active Server Pages with WebSpeed on page 223
Describes how to manage a WebSpeed installation to incorporate Microsoft Active Server Pages (ASP) in a WebSpeed application.

Part 5, Messaging and ESB Administration
OpenEdge Adapter for SonicMQ Administration on page 229
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.

Configuring and Managing the OpenEdge Adapter for Sonic ESB on page 241
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 6, REST Administration
REST Management Agent Administration on page 261
Describes the administrative tasks for the REST Management Agent, a component of Progress OpenEdge that you use to deploy, configure, and manage a Java container’s (or Web server’s) OpenEdge REST Web applications.

REST Web Application Administration on page 277
Describes the components, tools, and procedures for managing REST Web applications.

Using the RESTMAN Utility on page 303
Introduces the WSAMAN command-line utility to administer the REST Management Agent and the REST Web applications. The functions provided by the RESTMAN utility duplicate their OpenEdge Management and OpenEdge Explorer equivalents.

**REST Management Agent and REST Web Application Security Configurations on page 307**

Gives an overview of the security configurations provided by the REST Management Agent, describes the configuration models, and provides information on working with security configurations.

**Appendices**

**Reference to OpenEdge Web Service Properties on page 343**

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.

**Command and Utility Reference on page 357**

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.

**Reference to Dynamic Server Properties on page 421**

Lists the server properties that can be changed at runtime.

---

**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 runtime: "ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at runtime 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."

---

**Using ABL documentation**

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

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

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 documentation uses the following typographical and syntax conventions:

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

**Syntax:**

- **Fixed width**
  - A fixed-width font is used in syntax, code examples, system output, and file names.

- **Fixed-width italics**
  - Fixed-width italics indicate variables in syntax.

- **Fixed-width bold**
  - Fixed-width bold italic indicates variables in syntax with special emphasis.
### Convention

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE fixed width</td>
<td>ABL keywords in syntax and code examples are almost always shown in uppercase. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class.</td>
</tr>
<tr>
<td>Period (.) or colon (:)</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>[ ]</td>
<td>Large brackets indicate the items within them are optional.</td>
</tr>
<tr>
<td>[]</td>
<td>Small brackets are part of ABL.</td>
</tr>
<tr>
<td>{ }</td>
<td>Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.</td>
</tr>
<tr>
<td>{}</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></td>
<td></td>
</tr>
<tr>
<td>...</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**

```
ACCUM aggregate expression
```

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

```
FOR EACH Customer NO-LOCK:
    DISPLAY Customer.Name.
END.
```

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

**Syntax**

```
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**

```plaintext
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**

```plaintext
{ &argument-name }
```

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

**Syntax**

```plaintext
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**

```plaintext
MAXIMUM ( expression , expression [ , expression ] . . . )
```

In this example, you must specify MESSAGE and at least one `expression` or `SKIP [ ( n ) ]`, and any number of additional `expression` or `SKIP [ ( n ) ]` is allowed:

**Syntax**

```plaintext
MESSAGE { expression | SKIP [ ( n ) ] } . . .
```

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

---

OpenEdge Application Server: Administration
Syntax

{ include-file
  [ argument | &argument-name = "argument-value" ] ... }

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, WITH is followed by six optional items:

Syntax

WITH [ ACCUM max-length ] [ expression DOWN ]
  [ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]
  [ STREAM-IO ]

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, ASSIGN requires either one or more field entries or one record. Options available with field or record are grouped with braces and brackets:

Syntax

  | { record [ EXCEPT field . . . ] }

Example procedures

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

• A self-extracting Documentation and Samples file available on the OpenEdge download page of the Progress Software Download Center
Once installed, you can locate the example files in the following paths under the OpenEdge Documentation and Samples installation directory:

<table>
<thead>
<tr>
<th>This directory . . .</th>
<th>Contains examples for the following documents . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>src\prodoc\dotnetobjects</td>
<td>OpenEdge Development: GUI for .NET Programming</td>
</tr>
<tr>
<td>src\prodoc\dynamics</td>
<td>The Progress Dynamics documentation</td>
</tr>
<tr>
<td>src\prodoc\getstartoop</td>
<td>OpenEdge Development: Object-oriented Programming</td>
</tr>
<tr>
<td>src\prodoc\handbook</td>
<td>OpenEdge Getting Started: ABL Essentials</td>
</tr>
<tr>
<td>src\prodoc\interfaces</td>
<td>OpenEdge Development: Programming Interfaces</td>
</tr>
<tr>
<td>src\prodoc\json</td>
<td>OpenEdge Development: Working with JSON</td>
</tr>
<tr>
<td>src\prodoc\langref</td>
<td>OpenEdge Development: ABL Reference</td>
</tr>
<tr>
<td>src\prodoc\prodatasets</td>
<td>OpenEdge Development: ProDataSets</td>
</tr>
<tr>
<td>src\prodoc\tranman</td>
<td>OpenEdge Development: Translation Manager</td>
</tr>
<tr>
<td>src\prodoc\visualdesigner</td>
<td>OpenEdge Getting Started: Introducing Progress Developer Studio for OpenEdge Visual Designer</td>
</tr>
<tr>
<td>src\prodoc\xml</td>
<td>OpenEdge Development: Working with XML</td>
</tr>
<tr>
<td>src\samples\open4gl\java</td>
<td>OpenEdge Development: Java Open Client</td>
</tr>
</tbody>
</table>

**OpenEdge messages**

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

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

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

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

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

- Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.
• Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.

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

• Terminates the current session.

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

** Unknown table name table. (200)

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

Obtaining more information about OpenEdge messages

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

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

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

• In the Procedure Editor, press the HELP key or F1.

On UNIX platforms, use the OpenEdge pro command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

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

1. Start the Procedure Editor:

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

2. Press F3 to access the menu bar, then choose Help > Messages.

3. Type the message number and press ENTER. Details about that message number appear.

4. Press F4 to close the message, press F3 to access the Procedure Editor menu, and choose File > Exit.
Introduction

For details, see the following topics:

• 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.

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.

For details, see the following topics:

• AppServer for OpenEdge
• AppServer Internet Adapter
• OpenEdge Web Services Adapter
• WebSpeed Transaction Server and Messenger
• OpenEdge Adapter for SonicMQ
• OpenEdge Adapter for Sonic ESB
• REST Management Agent for REST Web application administration
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 Configuring and Managing the AppServer on page 39.

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 container, 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 Configuring and Managing the AppServer Internet Adapter on page 75.

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 container, which provides the Internet access that Web service clients require. However, in addition to the HTTP and HTTPS communications provided by the Java container/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 3, "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 4, "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 OpenEdge Adapter for SonicMQ Administration on page 229. 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 OpenEdge Web Services Adapter on page 32).

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 Configuring and Managing the OpenEdge Adapter for Sonic ESB on page 241.

Configuring the OpenEdge runtime container for Sonic

In a Sonic deployment scenario, you must perform the following steps to configure a runtime container:

1. Ensure that the Sonic Domain Manager is running.

2. To create an MF container on the computer on which OpenEdge is installed, execute $DLC/sonic/MQ8.0/bin/<hostnamecontainer>.bat, wherein <hostnamecontainer> is the batch file created during installation and hostname is the name of the computer on which OpenEdge is installed. Executing the batch file creates a runtime MF container and starts it. The MF container includes the Host Manager component; see the SonicMQ Configuration and Management Guide for information on the Host Manager.

3. Set the following properties for the MF container from the Sonic Management Console:
   
   - Select the Command Line check box in the General tab of the MF container properties section.
   
   - Set the value of Status poll interval to 120 in the Monitoring section of the Advanced tab of the MF container properties.
   
   - Set the value of Status poll timeout to 60 in the Monitoring section of the Advanced tab of the MF container properties.

4. Create an ESB container, preferably with the same name as the MF container, using the Sonic Management Console. For more information, refer to Progress Sonic Installation and Upgrade Guide.

5. Deploy the ESB container on the MF container.
Automatic upgrade of OpenEdge containers

In Sonic 8.0 and later, if the Sonic domain is upgraded, OpenEdge containers are automatically upgraded when the containers are restarted.

Note: OpenEdge Adapter for Sonic uses the Sonic launcher for setting up the runtime MF container and also for managing the lifecycle of the container (for example: to stop, start, and pause). For more information, refer to the Progress Sonic Installation and Upgrade Guide.

The OpenEdge Sonic JARs are associated with OpenEdge services instead of the ESB containers.

REST Management Agent for REST Web application administration

A REST Management Agent is a component of Progress OpenEdge that you use to deploy, configure, and manage a Java container’s (or Web server's) OpenEdge REST Web applications. Technically, the REST Management Agent is a Java Web application that acts as an intermediary for you to communicate with the deployed REST Web applications. It helps you enable or disable the application, collect statistics, and provide a runtime configuration for a REST Web application.

OpenEdge provides three clients—OpenEdge Management, OpenEdge Explorer, and RESTMAN utility—that use a REST Management Agent to manage the REST Web applications.

For more information on working with OpenEdge Management and OpenEdge Explorer to administer REST Management Agent and the REST Web applications, see the OpenEdge Management and OpenEdge Explorer: Configuration manual. For more information on REST Web application administration and on using RESTMAN utility, see Part 6, "REST Administration."
AppServer and Internet Adapter Administration

For details, see the following topics:

- Configuring and Managing the AppServer
- Configuring and Managing the AppServer Internet Adapter
Configuring and Managing the AppServer

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 Management or OpenEdge 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.

For details, see the following topics:

• Run-time components and operation
• AppServer administration framework
• Configuring AppServer components
• Setting up the environment for AppServer execution
• Starting and managing an AppServer instance
• Specifying the server pool parameters
• Managing code pages
• Checking for failed client connections
• Checking for failed server connections
• Summary of management tasks
Run-time components and operation

The following figure shows the major run-time components that compose the AppServer architecture and their basic relationships.

Figure 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.

The following table identifies and describes the components shown in the above figure.
<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
</table>
| Client application | A process that requests the execution of remote ABL procedures in the context of an AppServer session. A client application can be:  
• An ABL client session, including the WebClient  
• 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)  
• Another AppServer session—AppServer agent (for more information, see OpenEdge Application Server: Developing AppServer Applications)  
• A .NET Open Client application (for more information, see OpenEdge Development: .NET Open Clients)  
• A Java Open Client application (for more information, see OpenEdge Development: Java Open Clients)  
• A client of OpenEdge Web services (including Sonic ESB services)  
The OpenEdge interface in Figure 1: AppServer run-time components on page 40 is the code that allows a client application to access an AppServer. For ABL clients, this interface is accessed through built-in ABL statements and functions dedicated to AppServer access. For .NET, Java, and Web service Open Clients, this interface is accessed through a client interface that is custom-built with the OpenEdge Open Client Toolkit to access AppServer procedures. |
| AppServer agent    | 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 can execute in an interactive ABL client you can execute within an AppServer agent. An AppServer instance typically contains multiple AppServer agents that start up when you start the AppServer.  
An AppServer agent can also act as an ABL client of another AppServer instance, by making its own remote procedure calls. |
| Component        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |  |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|  |
| 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\(^1\) | 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). |  |

### 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.

---

\(^1\) 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.
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 Management or OpenEdge Explorer, 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 a 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 Management or OpenEdge Explorer, 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 - Security and Auditing*.

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 AppServer administration framework on page 47.

**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 SSL-enabled AppServer operation on page 45 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. The following figure shows an overview of this framework as it applies to AppServer administration.

**Figure 2: AppServer administration framework**

The above figure 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 Management or 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: AppServer administration framework on page 47):

- AppServer clients
- OpenEdge Management or 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 Management or 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 Management, OpenEdge Explorer, and the management utilities

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

- Use the browser-based OpenEdge Management or OpenEdge Explorer tool. OpenEdge Explorer is available if you do not install OpenEdge Management. If you install OpenEdge Management, it also includes the features of OpenEdge Explorer. OpenEdge Management provides access to the widest array of configuration settings controls for OpenEdge and external resources.
- 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 Management or OpenEdge Explorer

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

There are different options to consider when installing OpenEdge Management or OpenEdge Explorer. After completing installation (see OpenEdge Getting Started: Installation and Configuration), you can find complete documentation beginning with OpenEdge Management and OpenEdge Explorer: Getting Started or the OpenEdge Management and OpenEdge Explorer online help.

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 Management or 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 Management or OpenEdge Explorer, these utilities do not create a new AppServer configuration. Without OpenEdge Management or OpenEdge Explorer, you must use a text editor and the configuration utilities. For more information, see Text editor and configuration utilities on page 50.

This chapter describes how to manage an AppServer using OpenEdge Management or 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 on page 60.

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 *Configuring AppServer components* on page 51.

**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 *Configuring AppServer components* on page 51.

**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 *Editing the properties file* on page 55. 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:

- 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 Management or 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 the this utility and on updating the `ubroker.properties` file manually, see the Editing the properties file on page 55.

### 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 SSL-enabled AppServer operation on page 45 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 - Security and Auditing.

### Configuring AppServer components

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

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

To access OpenEdge Management or OpenEdge Explorer, open it from the OpenEdge program group. For more information, see OpenEdge Management and OpenEdge Explorer: Getting Started. Once you have opened OpenEdge Management or OpenEdge Explorer, you can configure an AppServer instance. The instructions below describe the general process. For specific instructions consult the online 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
Chapter 2: Configuring and Managing the AppServer

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 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 PROADSV on page 376.

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

2. In OpenEdge Management or OpenEdge Explorer, connect to the running AdminServer processes that you verified in 1 on page 51.

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 Management or OpenEdge Explorer online help.

**Configuring an AppServer with OpenEdge Management or OpenEdge Explorer**

You can use OpenEdge Management or OpenEdge Explorer to configure the AppServer on Windows or UNIX.

To configure an AppServer instance:

1. Make sure the AdminServer is running on the host where you want to configure the AppServer (see General steps for using OpenEdge Management or OpenEdge Explorer to configure an AppServer instance on page 51).

2. Start OpenEdge Management or OpenEdge Explorer using the appropriate startup procedure, access the AppServer installation you want to configure, then define a new AppServer instance or select an existing instance (see OpenEdge Management and OpenEdge Explorer: Getting Started, OpenEdge Management and OpenEdge Explorer: Configuration, or the online help).

3. Open the configuration pages for the AppServer instance, if they are not already open, and edit them. This displays the current instance settings for properties specified in the ubroker.properties file. These properties are displayed in separate groups, each of which is named for a particular property category. In OpenEdge Management or OpenEdge Explorer, these properties are listed with longer, more readable names that correspond to the actual names of properties in the properties file.

4. 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 set at least some properties under each category. See the online help for detailed information about each property.
The Broker category specifies properties of the AppServer broker. Selecting 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. You can also specify the name of the group (UNIX only).

- **Controlling NameServer** — You must indicate whether you plan to use a NameServer to control AppServer access. If so, check the Register With NameServer checkbox 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 OpenEdge Management or 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 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).

In addition, you can choose to allow the dynamic updating of properties and determine how the AppServer behaves when a client connection terminates. You can also set properties related to debugging and identify where you want code published from Progress Developer Studio for OpenEdge to be placed.

For more information on these options, see the OpenEdge Management or OpenEdge Explorer online help.

The Agent category specifies properties of the AppServer agents. Selecting 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 Customizing the AppServer agent executable on page 58.

- 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 Managing code pages on page 62.

• 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.)

• Specify how often data is flushed by the agent to the broker.

• **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); specify the maximum number of server log files to be kept (0 = unlimited number of log files retained); and specify the interval (in seconds) for the server log file watchdog thread. 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 Specifying the server pool parameters on page 62.

• **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 - Security and Auditing. Selecting this category 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 **Environment variable settings** on page 57.

---

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

---

5. **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 OpenEdge Management or OpenEdge 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 OpenEdge Management or 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 Management or 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*.

  **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 Reference to Dynamic Server Properties on page 421 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 the following table.

**Table 2: AppServer configuration entity names**

<table>
<thead>
<tr>
<th>Configuration entity name</th>
<th>Configuration entity function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[UBroker.AS.product-instance-name]</code></td>
<td>Defines property settings for a particular 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 AppServer product `[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 Management or 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:

**Table 3: Syntax**

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

For more information on the **ASCONFIG** utility, see **ASCONFIG** on page 370 and Summary of management tasks on page 71.

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 OpenEdge Management or OpenEdge Explorer. For more information, see Configuring AppServer components on page 51.
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 [Configuring AppServer components](#) on page 51.

**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 OpenEdge Management or OpenEdge 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 [Configuring AppServer components](#) on page 51.

**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—OpenEdge Management or OpenEdge 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 General steps for using OpenEdge Management or OpenEdge Explorer to configure an AppServer instance on page 51.

Starting and managing an AppServer with OpenEdge Management or OpenEdge Explorer

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

To access OpenEdge Management or OpenEdge Explorer, you can open them from the OpenEdge program group. For more options on each platform, see OpenEdge Management and OpenEdge Explorer: Getting Started.

If you want to learn more about using OpenEdge Management or OpenEdge Explorer, see OpenEdge Management and OpenEdge Explorer: Getting Started or the online help in the tool.

To start an AppServer instance:

1. In either OpenEdge Management or OpenEdge 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 *Specifying the server pool parameters* on page 62. For more information on AppServer status indicators, see *Table 5: AppServer agent status indications* on page 61.

- 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 *Specifying the server pool parameters* on page 62.

- Stop the AppServer.

Unlike OpenEdge Management or OpenEdge 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 **ubroker.properties** file or use OpenEdge Management or OpenEdge Explorer. For more information on managing log files, see *OpenEdge Getting Started: Installation and Configuration*. For more information on accessing the **ubroker.properties** file, see *Editing the properties file* on page 55.

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

### Table 4: Syntax

```
asbman {
  { -name AppServer-name
    { -kill | -start | -stop | -query |
      -addservers number-to-start
      -trimservers number-to-trim
      -listclients |
      -clientdetail connection-handle
      -listallprops |
      -refresh |
      -refreshStatus |
      -cancelRefresh }
  }
  [ -host host-name -user user-name] |
  [ -port port-number] |
} | -help }
```

For more information on the ASBMAN utility, see ASBMAN on page 362.

### Checking AppServer status

You can check the status of a running AppServer using either OpenEdge Management or OpenEdge 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 the following table.

### Table 5: 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 AppServers 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 Management or OpenEdge 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.

**Note:**

The `srvrStartupParam` property is a dynamic property. Any changes to this property affects only 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 (`RETURNstring`) 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.

Messages

ServerASK sends a message only 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.

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

Configuring clients

Clients that support ServerASK include:

- ABL client connections on page 66
- Java and .NET Open Client connections on page 66
- Web Services adapter connections on page 67
- Sonic Adapter connections on page 67
- SonicMQ Broker Adapter connections on page 67
- SonicESB Invocation Adapter connections on page 67
- SonicESB Web Services Adapter connections on page 68
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:

```apl
```

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.

Due to 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.

Checking for failed server connections

When a server 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 busy and application resources such as locked database records are unavailable to other users. It is important to detect these failures and recover resources as quickly as possible.

The AppServer provides the ClientASK messaging protocol to help identify failures. ASK stands for AppServer KeepAlive and ClientASK denotes that the direction of the keepalive messages is from clients to servers.

When an AppServer and client are configured to use ClientASK, the AppServer will periodically sends a keepalive message to the server and waits for a response. If the server 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 server 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 ClientASK protocol detects that a server bound to it is no longer connected as follows:

- The client infers that the server connection has failed. It does this, by keeping track of the time of the last activity on the connection.
If no messages are sent to, or are received from, the server within a specific time interval (specified in the `clientASKActivityTimeout` property), the client determines if the server is still connected.

The client sends an askPing request to the server.

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

When the client receives an askPing response, it updates the timestamp of the connection.

If the client does not receive a message from the server within a specific time interval (specified in the `clientASKResponseTimeout` property), the client will deem the connection to be in failure, and take action as if a fatal communication error has occurred. This may disconnect the socket.

Note that the client does not take action unless both the activity and response timers expire.

### Messages

`ClientASK` sends a message only after the interval set in the `clientASKActivityTimeout` property expires and no activity occurs between the server and client. Each successful `RunProcedure` request and response rests the timer. If the client issues a keepalive message and receives a keepalive response (or any other message from the server), 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 server.

**Note:** A client may be a newer or older software version than a server, so the `ClientASK` protocol establishes 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 a risk that a busy client may be slow in sending a keepalive request message to the AppServer. Careful tuning of the timer values is necessary to avoid disconnection of healthy client connections. Note that your client implementation details may add several seconds to the time it takes for the client to detect a timeout.

### AppServer operating modes

The `ClientASK` protocol can be used only for stateless and state-free operating modes of the AppServer. When operating in stateless and state-free modes, the client performs the `ClientASK` duties, since the client process remains connected to the AppServer in these modes.

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

### Configuring the AppServer

To use the `ClientASK` protocol, set the `appServerKeepaliveCapabilities` property in the `[UBroker.AS]` section of the `ubroker.properties` file to `allowClientAsk`.

The `appServerKeepaliveCapabilities` property contains the `ClientASK` protocol capabilities supported by this AppServer. Valid values are:

- `allowClientAsk`
- `denyClientAsk` (default)
Configuring clients

Clients that support ClientASK are:

- Java and .NET Open Client connections on page 70
- Web Services adapter and REST Web Services connections on page 70

Java and .NET Open Client connections

To request the use of the ClientASK protocol on a connection, a Java or .NET Open Client must set the new connection property prior to instantiation of the relevant 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 ClientASK protocol will be used on the connection. The value is either of the following:

- allowClientASK
- denyClientASK (default)

For more information on the working with Java and .NET Open Clients, see the OpenEdge Development: Java Open Clients and OpenEdge Development: .NET Open Clients manuals in the OpenEdge documentation set.

Web Services adapter and REST Web Services connections

To request the use of the ASK protocol for an application service, a Web Services adapter or the REST Web Service must set the following properties for that service, as described below:

- Set the appServerKeepalive property, a string valued property that indicates the ASK features, which the client would like to employ on the connection, to allowClientAsk. The appServerKeepalive property contains the ClientASK protocol capabilities supported by this client. The absence of this property indicates that the default value for the ClientASK protocol will be used on this connection. Valid values are:
  - allowClientAsk
  - denyClientAsk (default)

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

- Set the clientASKResponseTimeout property, an integer valued property that determines the maximum amount of time (in seconds) that the client waits for an askPing response after an askPing request is issued. The minimum value is 30. The default value is 60.

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

The following table describes the utilities that OpenEdge provides to help configure and manage an AppServer installation, listed generally in order by task.

Table 6: 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 Management or OpenEdge 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 Management or OpenEdge 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 NSMAN on page 374, PROADSV on page 376, and OpenEdge Getting Started: Installation and Configuration.

The Table 7: Management tasks on page 71 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 7: Management tasks

<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>To do this task . . .</td>
<td>Use these steps or commands . . .</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------</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 <code>ubroker.properties</code> 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(^2,(^3)</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(^2)</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(^2)</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 <code>ubroker.properties</code> 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>To do this task . . .</td>
<td>Use these steps or commands . . .</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Start a local AppServer</td>
<td><code>asbman -name AppServer-name -start</code></td>
</tr>
<tr>
<td>Start a remote AppServer(^2)</td>
<td><code>asbman -name AppServer-name -host host-name -port port-number -user user-name -start</code></td>
</tr>
<tr>
<td>Query a local AppServer</td>
<td><code>asbman -name AppServer-name -query</code></td>
</tr>
<tr>
<td>Query a remote AppServer(^2)</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(^2)</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(^2)</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>

\(^2\) Prompts for a password.
\(^3\) 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.

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.

For details, see the following topics:

- Installing the AppServer Internet Adapter
- Installing and configuring Web servers and Java container
- Configuring AIA components
- Viewing AppServer Internet Adapter connection and configuration information
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 container, with HTTPS enabled if you are using Secure Sockets Layer (SSL) connections
- A Web server with an integrated Java container, 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 Configuring AIA components on page 79. For information about installing and configuring Web servers, see Installing and configuring Web servers and Java container on page 77.

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 Java container 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. As an alternative, you can use OpenEdge Management or OpenEdge Explorer.
- The Java container or Web server must support server authentication. Supporting server authentication requires that X.509 digital certificates be installed on both the Web server (or Java container) 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 Enabling the Web server or Java container for SSL operation on page 79 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 Management or OpenEdge 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 Management or OpenEdge 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 - Security and Auditing.

• 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 container

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

• Add OpenEdge-Install-Directory/java/aia.zip to the Java container CLASSPATH.

• On UNIX only, add OpenEdge-Install-Directory/lib to the Java container 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 Java container:

  - Add:


    \[
    \text{DLC=} \text{OpenEdge-Install-Directory}; \text{ export DLC}
    \]

  - Add:


    \[
    \text{WRKDIR=} \text{OpenEdge-working-directory}; \text{ export WRKDIR}
    \]

**Note:** This is the path to the work directory you specified when you installed OpenEdge.

  - Add:


    \[
    \text{LD_LIBRARY_PATH = OpenEdge-Install-Directory/lib:}$LD_LIBRARY_PATH; \text{ export LD_LIBRARY_PATH}
    \]

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

You must restart your Java container 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:

**Table 8: Syntax**

\[
\text{http://Host:Port/Path}
\]

- **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 Java container and the AIA and must include the following information:

  - A virtual path that the Java container has configured within the Web server to recognize and pass directly to the Java container instead of looking for that directory on the Web server
  
  - Name of the Java servlet to be invoked by the Java container
The specification of the Path depends on your Web server and Java container. For more information, see Configuring AIA components on page 79 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 Java container, see the documentation for that Java container. For more specific information about configuring the AIA with Java container, 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 Java container for SSL operation**

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

To secure the Web server or Java container:

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 - Security and Auditing and Security considerations for AIA administration on page 76.

**Configuring AIA components**

You must configure AIA for your specific deployment. A running AIA instance is an extension of the Java container, and the Java container starts and stops the AIA instance when necessary. When the Java container 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 Java container.

This section describes how to configure an AIA instance by using OpenEdge Management or OpenEdge Explorer and by editing the `ubroker.properties` file.
General steps for using OpenEdge Management or OpenEdge Explorer to configure an AIA instance

Access OpenEdge Management or OpenEdge 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 AIA as 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 PROADSV on page 376.

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

2. In OpenEdge Management or OpenEdge 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 Configuring an AIA with OpenEdge Management or OpenEdge Explorer on page 80.

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

Configuring an AIA with OpenEdge Management or OpenEdge Explorer

To configure the AIA in Windows, you can use OpenEdge Management or 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 General steps for using OpenEdge Management or OpenEdge Explorer to configure an AIA instance on page 80).

2. Open OpenEdge Management or 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, choose Resources® New OpenEdge Resource® AppServer Internet Adapter. Then click the Help icon (question mark) for details about creating a new adapter.
   - To modify an existing AIA configuration, expand the AppServer Internet Adapter folder in the list frame, select the AIA instance you want to modify, and click Configuration in the detail frame on the right.

5. Click the tab for each property category. You can accept the default values, if they are appropriate for your application. Be sure to review and set properties in 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 Connect using NameServer box and select a controlling NameServer from the list of Controlling NameServer instances that you have already configured; also, for the NameServer client port fields, enter a minimum and a maximum value (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, provide the AppServer host name and AppServer port you want 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 OpenEdge Management or 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.
   - 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.

To allow the AppServer to recognize that a client bound to it is no longer connected, select the AppServer KeepAlive box. To allow dynamic property updates, select the Enable dynamic property updates box.
Configuring an AIA by editing the properties file

Although it is recommended that you use OpenEdge Management or OpenEdge Explorer whenever possible, as an alternative you can edit the ubroker.properties file to configure the AIA. See Editing the properties file on page 55 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 the following table.

Table 9: 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 ubroker.properties file can contain several of these entities, each with a unique product-instance-name. For an AIA, product-instance-name 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 Management or 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:

Table 10: Syntax

```
```

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 AIACONFIG on page 361.
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:

**Table 11: Syntax**

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

For more information on this URL, see Installing and configuring Web servers and Java container on page 77.

**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 Java container.

The `Path` used to view connection and configuration information depends on how you configure your Web server and Java container. 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 Java container. 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
```

The following figure shows connection information using the above URL.

**Figure 3: AIA client connection information**

![AIA client connection information](image)
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
```

The following figure shows AIA configuration information for the previous sample URL.

**Figure 4: AIA configuration information**

![Summary of Connection Status]

### Summary of Connection Status

<table>
<thead>
<tr>
<th>ID</th>
<th>User Info</th>
<th>AppSvc</th>
<th>Host:Port</th>
<th>#Pkt Sent</th>
<th>#Pkt Revd</th>
<th>Connected</th>
<th>Last Accessed</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>None</td>
<td>ubroker1</td>
<td>172.18.2.177:2022</td>
<td>000531</td>
<td>000531</td>
<td>28-Jul-00 16:47</td>
<td>28-Jul-00 16:49</td>
</tr>
</tbody>
</table>

### Configuration Information for AiaJRun30

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Current Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation Directory</td>
<td>/usr2/http诰/progress/91b</td>
</tr>
<tr>
<td>HTTPS Enabled</td>
<td>False</td>
</tr>
<tr>
<td>Log Filename</td>
<td>/usr2/http诰/progress/ork/AiaJRun30.ain.log</td>
</tr>
<tr>
<td>Logging Level</td>
<td>2</td>
</tr>
<tr>
<td>Log Append</td>
<td>True</td>
</tr>
<tr>
<td>Controlling NameServer</td>
<td>NS1</td>
</tr>
<tr>
<td>NameServer Host:Port</td>
<td>localhost:5162</td>
</tr>
<tr>
<td>Time interval before trimming connection (seconds)</td>
<td>3600</td>
</tr>
<tr>
<td>Secured Port</td>
<td>0</td>
</tr>
<tr>
<td>NameServer Client Port</td>
<td>-1</td>
</tr>
</tbody>
</table>

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.
Web Services Adapter Administration

For details, see the following topics:

- Configuring a Web Services Adapter Installation
- Managing the Web Services Adapter
- Deploying and Managing OpenEdge Web Services
- Web Services Adapter Security Configurations
- 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 container 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 Java container Web application. Here, you can specify parameters that affect the Java container 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.

For details, see the following topics:

• Moving the WSA sample Web application
• Configuring the Java container to recognize the WSA
• Tuning the Java container 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 container 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 the following table.

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

3. Update your Java container configuration files to refer to the new WSA sample Web application directory. For more information, see Configuring the Java container to recognize the WSA on page 90. In general, you need to restart the Java container (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 3 on page 90. 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 Deploying and Managing OpenEdge Web Services on page 107.

Configuring the Java container to recognize the WSA

The WSA component runs as a Web application within a Java container. Therefore, you must inform the Java container 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 Moving the WSA sample Web application on page 90). To configure the Java container to recognize the WSA component, and for more information on installing Web applications from an expanded source-directory tree, see your Java container documentation. Otherwise, for information on configuring the WSA as a Web application, see Managing the Web Services Adapter on page 93.
Tuning the Java container 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 Java container 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 `-Xmxsize` 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 Web Services Adapter Security Configurations on page 117.

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 Security considerations for Web service administration on page 108.

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 Security considerations for Web service administration on page 108 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 WSA administration architecture on page 94.

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 Java container.
2. Start the WSA the Web server's Java container 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 Management/OpenEdge Explorer or the `WSAMAN` utility by specifying the remote WSA's URL. For details, see the OpenEdge Management/OpenEdge Explorer online help.

   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 Management/OpenEdge Explorer or the `WSAMAN` utility to manage the "remote" WSA's dynamic run-time properties and deploy and manage the WSA's SOAP services.

OpenEdge Management or 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 Management or 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 container. You can manage the WSA using the Unified Broker framework, including OpenEdge Management/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.

The WSA includes extensive security features, which this chapter introduces, but for more detailed information on WSA security, see Web Services Adapter Security Configurations on page 117.

For details, see the following topics:

- Customizing the Timeout parameter for Web Service Adapter (WSA)
- 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
Customizing the Timeout parameter for Web Service Adapter (WSA)

You can now set the default timeout value for pinging the Web Service Adapter (WSA) using an environment variable. In the `PluginPolicy.Progress.AdminServer` section, set a value for the `-Dpsc.httpTimeout` parameter (in seconds) in the `$DLC/properties/AdminServerPlugins.properties` file as in the example below:

```java
jvmargs= -Xmx384m -XX:MaxPermSize=128m -Djava.awt.headless=true
-Dsun.lang.ClassLoader.allowArraySyntax=true -Dpsc.httpTimeout=2
```

WSA administration architecture

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

- WSA administration prerequisites on page 94
- Web server and Java container configurations on page 95
- Constructing URLs on page 96
- WSA as a Java container Web application on page 98

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 Management/OpenEdge Explorer or the 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 Java container 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.
The following figure shows the components used to configure and manage a WSA in its environment.

**Figure 5: WSA administration architecture**

Web server and Java container configurations

As shown in Figure 5: WSA administration architecture on page 95, the WSA is implemented as a Java servlet that runs in the context of a Java container as a Web application. The Java container 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 Java container or Web server, several WSA instances can run within the context of the WSA Web application (such as `wsa1` and `wsa2` shown in the above figure). In addition to the WSA Web application other Web applications can run within the Java container, including additional WSA instances.

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

```plaintext
Web server (root)  
  Java container (web-server-context)  
    WSA Web application (wsa-webapp-context)  
      WSA instance (wsa-instance)  
        ...  
        Non-WSA Web application  
        ...  
        ...  
        Non-Java container 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 Java containers (web-server-context) and one or more non-Java container Web resources, such as static HTML pages.
3. Each Java container comprises one or more WSA Web applications (wsa-webapp-context) and one or more non-WSA Web applications.

4. Each WSA Web application comprises one or more WSA instances (wsa-instance).

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

<table>
<thead>
<tr>
<th>Stand-alone Java container (root)</th>
<th>WSA Web application (wsa-webapp-context)</th>
<th>WSA instance (wsa-instance)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>Non-WSA Web application</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

In both these configurations, web-server-context, wsa-webapp-context, and 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 wsa-root-url. When you install the WSA, the installation procedure prompts you for a wsa-root-url value for the sample WSA instance (wsa1). Thereafter, every time you create a new WSA instance you must provide a new wsa-root-url value for that instance. The value is stored in the wsaUrl property in ubroker.properties.

The wsa-root-url has the following syntax:

**Table 13: Syntax**

```
http[s]://host:port[/web-server-context]/wsa-webapp-context/wsa-instance
```

**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, /web-server-context, if the WSA runs in a Java container that runs in a separate Web server. Omit the optional element if the WSA runs in a Java container 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 the following table.

**Table 14: URL components**

<table>
<thead>
<tr>
<th>This URL component . . .</th>
<th>Which is specified in . . .</th>
<th>Names . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>web-server-context</td>
<td>The Web server configuration file</td>
<td>The connection between the Java container and the Web server (if any)</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 the following table.

**Table 15: 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 the above table, the following are true:

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

  http://servicehost:80/bedrock/quarry/fred

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

  http://servicehost:8080/quarry/fred

Any URL subpaths specified by the client that lie beyond `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 `wsa-root-url`, as shown in bold:

http://servicehost:80/bedrock/quarry/fred/wsdl

Given this URL, an HTTP message travels through the Web server, through the Java container, 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 Java container Web application

To configure a WSA, it is also necessary to understand the architecture of the WSA as a Java container Web application. The WSA Web application consists of a set of directories and files. The following table shows the directory structure.

Table 16: Directory structure of the WSA as a Java container Web application

```
./WSA-web-application-directory
  [ optional static html pages ]
  .
  .
  WEB-INF/
    web.xml ( WSA Web application descriptor file )
    lib/
      wsa.jar
      soap.jar
    [ any additional WSA Web application jar files ]
    .
    .
    classes/
      [ optional Java .class files ]
      .
      .
```

The WSA Web application consists of a subdirectory named WEB-INF (the name is case sensitive) that contains the Web application descriptor file (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 Java container can contain multiple WSA Web applications, and each Web application can contain multiple WSA instances.

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

Defining a Web application for the WSA

You must define a Web application to allow the Java container 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 Configuring a Web Services Adapter Installation on page 89.

2. Configure the Java container to recognize the WSA Web application. Depending upon the Java container, 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 Java container typically has a directory where a Web application resides. See your Java container 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 Java container. For more information, see your Java container vendor's documentation. The WSA comes preconfigured to use the role definitions PSCAdmin and PSCOper (case-sensitive). With security enabled, the WSA requires the Java container to authenticate and authorize a user before passing the HTTP request to the Java container for handling. For more information on WSA security, see Web Services Adapter Security Configurations on page 117.

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 Management or OpenEdge Explorer to create a WSA instance.

To create a WSA instance:

1. Choose one:
   - From the management console bar, click the Resources tab's dropdown arrow. Choose New OpenEdge Resource—>WebServices Adapter.

   The WebServices Adapter Configuration page appears.

2. Type the name of the new Web Services Adapter in the New Web Services name field.

   Note: The Web Services Adapter name is case sensitive and can include any character except a period (.) or square brackets ([ ]). The name must be unique among all configured NameServer names.

3. Type the URL in the Url field.

   This is a URL that directly addresses the WSA instance on the Internet and serves as the root for other URLs required to access Web services and other functions of the WSA.

   Form the root URL according to the following format:
http://host[:port][/java-container-context]/webapp-context/servlet-instance

Where:

- **host** — The name of the machine where the WSA is installed (Default: localhost).
- **port** — The HTTP port number on the WSA machine (Default: 80).
- **java-container-context** — The Java container context path name is present only when you have a Web server between your Java container and the client application. It specifies the path name that you define in your Web server configuration to send client requests to your Java container.
- **webapp-context** — The Web application context path name always exists and specifies the path name that you define in your Java container configuration to identify the WSA as a servlet application.
- **servlet-instance** — The servlet-mapping name for the WSA instance (WSA Web application) specified in OpenEdge-install-dir/servlets/wsa/WEB-INF/web.xml. This is also the name by which the WSA instance appears in the Web Services Adapter folder in the management console.

4. Select the WSA installation location for this instance from these choices:
   - **local** — The WSA is installed on the machine where the selected AdminServer is running.
   - **remote** — The WSA is installed on a network machine that is remote from the selected AdminServer.

5. Click **Save**. The new WSA instance appears under the **Web Services Adapter** folder.

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

7. 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 (wsal) to a new section—in essence, cloning wsal's section to a new section.
   b) Then, edit the properties in the new WSA instance's section as desired.

8. In the web.xml file, use a text editor (or whatever tool your Java container provides), to copy and rename the section corresponding to the sample WSA instance provided (wsal) 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 the following table.
Table 17: 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></td>
<td></td>
</tr>
<tr>
<td>InstallDir</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;param-value&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;init-param&gt;</td>
<td>The name of the WSA instance</td>
<td>wsal</td>
</tr>
<tr>
<td>&lt;param-name&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>instanceName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;param-value&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;init-param&gt;</td>
<td>The location of the ubroker.properties file</td>
<td>C:\Progress\OpenEdge\</td>
</tr>
<tr>
<td>&lt;param-name&gt;</td>
<td></td>
<td>properties\ubroker.properties</td>
</tr>
<tr>
<td>propertyFileName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;param-value&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;init-param&gt;</td>
<td>Note: It need not reside in InstallDir</td>
<td></td>
</tr>
<tr>
<td>&lt;param-name&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>propertyFileName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;param-value&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;init-param&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;param-name&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>propertyFileName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;param-value&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;servlet-mapping&gt;</td>
<td>The name of the WSA servlet</td>
<td>wsal_servlet</td>
</tr>
<tr>
<td>&lt;servlet-name&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;servlet-mapping&gt;</td>
<td>The URL pattern of the WSA servlet</td>
<td>/wsal/*</td>
</tr>
<tr>
<td>&lt;url-pattern&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;security-constraint&gt;</td>
<td>The role-name of the default WSA administrator</td>
<td>PSCAdmin</td>
</tr>
<tr>
<td>&lt;auth-constraint&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;role-name&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;security-constraint&gt;</td>
<td>The role-name of the default WSA operator</td>
<td>PSCOpr</td>
</tr>
<tr>
<td>&lt;auth-constraint&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;role-name&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;security-constraint&gt;</td>
<td>The type of connection for accessing WSA administration, WSDL, and Web service</td>
<td>CONFIDENTIAL (if you are using SSL)NONE (if you are not using SSL)</td>
</tr>
<tr>
<td>&lt;user-data-constraint&gt;</td>
<td>applications</td>
<td></td>
</tr>
<tr>
<td>&lt;transport-guarantee&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 Web Services Adapter Security Configurations on page 117.

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 Java container is running.
2. Following the instructions for your specific Java container, start the WSA as a Web application. See your Java container vendor documentation on starting and running a Web application.

   Note: Depending on the behavior of a specific Java container, starting the Java container 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 Java container 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 Management or OpenEdge Explorer, or by manually editing the file.

The following table 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 Management or OpenEdge Explorer online help.
### Table 18: ubroker.properties properties of a WSA instance

<table>
<thead>
<tr>
<th>Category</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Web Services Adapter Location URL Web Service Protocol Type</td>
</tr>
<tr>
<td>Proxy Server Setting</td>
<td>Host Name Port Number Username Password</td>
</tr>
<tr>
<td>WSDL</td>
<td>Enable WSDL Retrieval WSDL Inaccessible Page Enable WSDL Listing Retrieval WSDL Listing Page</td>
</tr>
<tr>
<td>Logging</td>
<td>Working Directory Log Filename Web Services Adapter Logging Level Append to Web Services Adapter Log File Log Message Threshold Logging entry types Log file threshold Maximum number of files</td>
</tr>
<tr>
<td>Advanced Features</td>
<td>Admin SOAP Action Header HTTP Error Page Debug Clients Client Host</td>
</tr>
</tbody>
</table>
Modifying properties of a WSA instance

To modify most startup properties of a WSA instance, you must stop and start the Java container associated with the WSA instance. However, the following WSA properties can be modified at run time (while the WSA is running):

- debugClients
- enableWsdl
- enableWsdlListings
- logLevel
- 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 Management/OpenEdge Explorer or the WSAMAN utility. To modify the startup properties of a WSA instance, use OpenEdge Management or OpenEdge Explorer.

For more information on OpenEdge Management or OpenEdge Explorer, see the online help. For more information on the WSAMAN utility, see Using the WSA Management Utility (WSAMAN) on page 141 and Command and Utility Reference on page 357.

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:

**Table 19: Syntax**

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

For more information on the WSACONFIG utility, see WSACONFIG on page 395.
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:

```
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. The following table lists the statistics of a WSA instance.

**Table 20: 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>
<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>

**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 Deploying and Managing OpenEdge Web Services on page 107.
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.

For information on using ProxyGen to generate a WSM file for Web service deployment, see OpenEdge Development: Open Client Introduction and Programming.

For details, see the following topics:

- 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
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 Management/OpenEdge Explorer or the WSAMAN utility. The following table lists some of these default Web services properties.

Table 21: 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>
<tbody>
<tr>
<td>appServiceProtocol</td>
<td>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).</td>
<td>AppServer</td>
</tr>
<tr>
<td>appServiceHost</td>
<td>The name of the NameServer system or AppServer system, depending on the setting of appServiceProtocol.</td>
<td>localhost</td>
</tr>
<tr>
<td>appServicePort</td>
<td>The TCP/IP port number, which depends on whether the appServiceHost property specifies a NameServer or AppServer system.</td>
<td>5162</td>
</tr>
<tr>
<td>appServiceName</td>
<td>The name of the default AppServer service.</td>
<td>asbroker1</td>
</tr>
<tr>
<td>serviceLoggingLevel</td>
<td>A value between 1 and 4 to identify the logging level for Web services applications. The greater the value, the more logging occurs.</td>
<td>2</td>
</tr>
<tr>
<td>serviceAvailable</td>
<td>1 to allow client access to enabled Web service applications. Set to 0 to prevent client access.</td>
<td>0</td>
</tr>
</tbody>
</table>

For more information on modifying the default Web service properties of a WSA instance, including a full list of default properties, see the OpenEdge Management or OpenEdge Explorer online help, or see Using the WSA Management Utility (WSAMAN) on page 141, and Command and Utility Reference on page 357.

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 Enabling the WSA for HTTPS client connections on page 91 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 Java container 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 Management/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 Web Services Adapter Security Configurations on page 117)
- 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 Management/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 SSL-enabled AppServer operation on page 45 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 - Security and Auditing.

**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 Reference to OpenEdge Web Service Properties on page 343.

**Deploying a Web service**

You can use OpenEdge Management/OpenEdge Explorer or the WSAMAN utility's deploy function to deploy a Web service. This section describes deploying a Web service with OpenEdge Management or OpenEdge Explorer. For more information on using WSAMAN to deploy a Web service, see Using the WSA Management Utility (WSAMAN) on page 141 and Command and Utility Reference on page 357.
See *OpenEdge Management and OpenEdge Explorer: Configuration* for complete information on deploying a web service using OpenEdge Management or 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, the following table shows how you might represent the first version of the Web service.

<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>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>

The following table shows how you might represent the second version of the Web service.

<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>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, you will notice in OpenEdge Management or OpenEdge Explorer an icon for the Web services under the **Web Services** folder for the WSA instance. Also, the files listed in the following table are created in the WSA instance directory.
Table 24: 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.wsdl</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 Management/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 Management or OpenEdge Explorer online help, Using the WSA Management Utility (WSAMAN) on page 141, Command and Utility Reference on page 357, and Reference to OpenEdge Web Service Properties on page 343.

**Changing the Web service deployment directory**

As described in Configuring a Web Services Adapter Installation on page 89 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 the Managing the Web Services Adapter on page 93 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:

   ```xml
   <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:

   ```bash
   OpenEdge-Install-Directory/servlets/wsa/ws1/* C:/work/deployment/ws1/*
   ```

6. Restart the Java container 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 Management/OpenEdge Explorer or WSAMAN utility's enable function. This section describes how to enable a Web service using OpenEdge Management or OpenEdge Explorer. For more information on using the WSAMAN utility to enable a Web service, see the Using the WSA Management Utility (WSAMAN) on page 141 and Command and Utility Reference on page 357

See OpenEdge Management and OpenEdge Explorer: Configuration for complete information on enabling a Web service using OpenEdge Management or OpenEdge Explorer. You can also access this information from the OpenEdge Management or OpenEdge Explorer 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 the following table.

Table 25: 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:</td>
<td>Export creates a Web Service Definition (WSD) file; import requires an existing WSD file.</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 Management/OpenEdge Explorer or the WSAMAN utility. For more information, see the online help, Using the WSA Management Utility (WSAMAN) on page 141 and Command and Utility Reference on page 357.

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.

**Note:** 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 Management/OpenEdge Explorer or the **WSAMAN** utility runs.

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

For more information on exporting and importing Web services using the **WSAMAN** utility, see Using the WSA Management Utility (WSAMAN) on page 141 and Command and Utility Reference on page 357.

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 OpenEdge Management/OpenEdge Explorer or the **WSAMAN** utility. The following table lists the types of statistics gathered for a Web service.

**Table 26: Statistics for a Web service**

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of Requests</strong></td>
<td>Total requests handled by the Web service</td>
</tr>
<tr>
<td><strong>Number of Active Requests</strong></td>
<td>Requests currently being serviced</td>
</tr>
<tr>
<td><strong>Number of SOA Faults</strong></td>
<td>Error returns by the Web service application</td>
</tr>
<tr>
<td><strong>Number of Objects Not Found</strong></td>
<td>Calls to unidentified Web service methods</td>
</tr>
<tr>
<td><strong>Number of Errors</strong></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><strong>Number of AppServer Connections</strong></td>
<td>Total AppServer connections being managed by this Web service</td>
</tr>
<tr>
<td><strong>Number of Proxy AppObjects</strong></td>
<td>Total AppObjects registered for all clients of this Web service</td>
</tr>
<tr>
<td><strong>Number of Proxy SubAppObjects</strong></td>
<td>Total SubAppObjects registered for all clients of this Web service</td>
</tr>
<tr>
<td><strong>Number of Proxy ProcObjects</strong></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 the OpenEdge Management or 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 Reference to OpenEdge Web Service Properties on page 343 and the OpenEdge Management or 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

**Deploying in a production 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.

**Deploying in a development environment**

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 Management/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.
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.

For details, see the following topics:

- Overview
- Initial settings
- Alphabetical list of configuration instructions

**Overview**

When WSA security is enabled, the Java container 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 Java container**

The WSA's `web.xml` file can be set to tell the Java container that one or more of its URL paths require user authentication. When this is set, the Java container 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 Java container intercepts an HTTP request destined for the WSA instance's Web service application URL:

1. The Java container performs HTTP Basic authentication using the user's ID and password.
2. If the user ID and password are valid, the Java container checks that the user ID has been granted access to the Role WsaWebServicesUser.
3. If the user ID has been granted access, the Java container 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 Java container 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 Java container:

1. The WSA first checks its ubroker.properties values to determine whether the URL requires the Java container to perform user authentication.
2. If it does, the WSA verifies that the Java container has passed it a valid user ID—just in case the Java container 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.

Java container and WSA security files

Java container security and WSA security involve different files. Java container 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 Java container 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 Java container 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 Java container provides user authentication and role authorization. To these security features, the WSA adds the following, all of which are optional:

- Requiring the Java container 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

Java container security is initially set in OpenEdge-Install-Directory/servlets/wsa/web.xml as shown in the following table.

Table 27: Initial settings for Java container 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/wsdl, 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 the following table.
### Table 28: 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` file (`OpenEdge-Install-Directory/servlets/wsa`) 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:

```xml
<!--
   <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:

- **Using user-authorization role-names** on page 122
- **Using user-authorization role-names** on page 122
- **Customizing the default administrator's permissions and action settings** on page 124
- **Customizing the PSCOper role's permissions and action settings** on page 124
- **Customizing WSA administration roles** on page 125
- **Disabling access to all Web services, to all WSDL, or to all WSA administration** on page 127
- **Disabling authentication for WSA administration** on page 127
- **Disabling the deploying and undeploying of Web services** on page 128
- **Disabling WSDL listings** on page 128
- **Enabling authentication for Web services** on page 129
- **Enabling authentication for WSDL** on page 130
- **Enabling multiple user roles to access Web services, WSDL, or WSA administration** on page 132
- **Enabling Web services per user and per application** on page 134

**Controlling access to Web services, WSDL, and WSA administration**

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 the following:

- **Using one Java container security-constraint** on page 121
- **Using user-authorization role-names** on page 122

**Using one Java container security-constraint**

The WSA provides three different Java container 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 the following table.
### Table 29: Controlling Web service, WSDL, and administration access using Java container security constraints

<table>
<thead>
<tr>
<th>For accessing . . .</th>
<th>The security-constraint might look like this . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Web services</strong></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><strong>WSDL</strong></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><strong>WSA administration</strong></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>

### 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:

   ```xml
   <role-name>name</role-name>
   ```

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 `PSCAdmin` and `GuestAdmin`, you might modify the security-constraints as shown in the following table.
Table 31: 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>
| Web services       | <security-constraint>
|                    |   <web-resource-collection>
|                    |     <url-pattern>/wsa1/</url-pattern>
|                    |     <auth-constraint>
|                    |       <role-name>PSCAdmin</role-name>
|                    |       <role-name>GuestAdmin</role-name>
|                    |     </auth-constraint>
|                    |   </web-resource-collection>
|                    | </security-constraint>                                       |
| WSDL               | <security-constraint>
|                    |   <web-resource-collection>
|                    |     <url-pattern>/wsa1/wSDL/</url-pattern>/</url-pattern>
|                    |     <auth-constraint>
|                    |       <role-name>PSCAdmin</role-name>
|                    |       <role-name>GuestAdmin</role-name>
|                    |     </auth-constraint>
|                    |   </web-resource-collection>
|                    | </security-constraint>                                       |
| WSA administration | <security-constraint>
|                    |   <web-resource-collection>
|                    |     <url-pattern>/wsa1/admin/*</url-pattern>/*</url-pattern>*/
|                    |     <auth-constraint>
|                    |       <role-name>PSCAdmin</role-name>
|                    |       <role-name>GuestAdmin</role-name>
|                    |     </auth-constraint>
|                    |   </web-resource-collection>
|                    | </security-constraint>                                       |

3. Modify the properties of the WSA instance to require Java container 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, select the WSA instance in OpenEdge Management or OpenEdge Explorer, select Configuration, and select the Security tab.

Then, click Edit and enable the check boxes to require Java container 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 Java container authentication of all users of that function, as shown in the following table.
Table 32: Requiring Web service user authorization for Web service, WSDL, and administration access

<table>
<thead>
<tr>
<th>To require Java container 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 Java container 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 PSCOOper 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 PSCOOper 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:

```properties
[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 Java container 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:

```xml
<security-constraint>
  <web-resource-collection>
    <url-pattern>/wsal/admin/*</url-pattern/>
  </web-resource-collection>
</security-constraint>
```

The syntax for a role-name element is:

```xml
<auth-constraint>
  <role-name>name</role-name>
</auth-constraint>
```

Table 33: Syntax
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:

```xml
<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>
```

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:

```properties
[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,delete
  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, select the WSA instance in OpenEdge Management or OpenEdge Explorer. Then, in the WSA's Security tab, in the Admin Role Links, add the new role-names. For more information, see the online help.

   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:

```properties
[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, select the WSA instance in OpenEdge Management or OpenEdge Explorer 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 Using user-authorization role-names on page 122. You can also refer to the OpenEdge Management or OpenEdge Explorer online help.

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 the following table.

Table 34: 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>webAppEnabled</td>
<td>0</td>
</tr>
<tr>
<td>WSDL</td>
<td>enableWsd1</td>
<td>0</td>
</tr>
<tr>
<td>WSA administration</td>
<td>adminEnabled</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 Java container 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 Management or 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), select the WSA instance in OpenEdge Management or OpenEdge Explorer, and click Configuration. Select Security to display the Security tab. Click Edit, and then 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=0
   ```

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) Using OpenEdge Management or 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, select the WSA instance in OpenEdge Management or OpenEdge Explorer, click Configuration, and then click 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:

```plaintext
[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:

   ```xml
   <!--
   <security-constraint>
   <web-resource-collection>
   <url-pattern>/wsa1/</url-pattern>
   </web-resource-collection>
   </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
   <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 Java container 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), select the WSA instance
     in OpenEdge Management or OpenEdge Explorer, 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 the following table.

   **Table 36: 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:

   ```
   <security-constraint>
   <web-resource-collection>
     <url-pattern>/wsa1/wsdl/*</url-pattern>
   </web-resource-collection>
   </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:

   For example, after you add the role-names wsdlrole1 and wsdlrole2, the <security-constraint> element might appear as follows:

   ```
   <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 Java container 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), select the WSA instance in OpenEdge Management or OpenEdge Explorer, 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 wsd1Auth to 1 (one). The possible values for wsd1Auth are listed in the following table.
Table 38: 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 the following table.
Table 39: 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><code>&lt;security-constraint&gt;</code>&lt;br&gt;<code>&lt;web-resource-collection&gt;</code>&lt;br&gt;<code>&lt;url-pattern&gt;</code>/wsa1&lt;/url-pattern&gt;&lt;br&gt;<code>&lt;/web-resource-collection&gt;</code>&lt;br&gt;<code>&lt;/security-constraint&gt;</code></td>
</tr>
<tr>
<td>WSDL</td>
<td><code>&lt;security-constraint&gt;</code>&lt;br&gt;<code>&lt;web-resource-collection&gt;</code>&lt;br&gt;<code>&lt;url-pattern&gt;</code>/wsa1/wsd1/*&lt;/url-pattern&gt;&lt;br&gt;<code>&lt;/web-resource-collection&gt;</code>&lt;br&gt;<code>&lt;/security-constraint&gt;</code></td>
</tr>
<tr>
<td>WSA administration</td>
<td><code>&lt;security-constraint&gt;</code>&lt;br&gt;<code>&lt;web-resource-collection&gt;</code>&lt;br&gt;<code>&lt;url-pattern&gt;</code>/wsa1/admin/<em>&lt;/url-pattern&gt;/</em>&lt;br&gt;<code>&lt;/web-resource-collection&gt;</code>&lt;br&gt;<code>&lt;/security-constraint&gt;</code></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:

**Table 40: 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 the following table.
Table 41: Setting security constraints for multiple user roles on Web services, WSDL, and administration

<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><code>&lt;security-constraint&gt;</code>&lt;br&gt;<code>&lt;web-resource-collection&gt;</code> &lt;br&gt;<code>&lt;url-pattern&gt;/wsa1/&lt;/url-pattern&gt;</code>&lt;br&gt;<code>&lt;auth-constraint&gt;</code>&lt;br&gt;<code>&lt;role-name&gt;WSUser1&lt;/role-name&gt;</code>&lt;br&gt;<code>&lt;role-name&gt;WSUser2&lt;/role-name&gt;</code>&lt;br&gt;<code>&lt;/auth-constraint&gt;</code>&lt;br&gt;<code>&lt;/web-resource-collection&gt;</code>&lt;br&gt;<code>&lt;/security-constraint&gt;</code></td>
</tr>
<tr>
<td>WSDL</td>
<td><code>&lt;security-constraint&gt;</code>&lt;br&gt;<code>&lt;web-resource-collection&gt;</code> &lt;br&gt;<code>&lt;url-pattern&gt;/wsa1/wsd1/*/&lt;/url-pattern&gt;</code>&lt;br&gt;<code>&lt;auth-constraint&gt;</code>&lt;br&gt;<code>&lt;role-name&gt;WSDLUser1&lt;/role-name&gt;</code>&lt;br&gt;<code>&lt;role-name&gt;WSDLUser2&lt;/role-name&gt;</code>&lt;br&gt;<code>&lt;/auth-constraint&gt;</code>&lt;br&gt;<code>&lt;/web-resource-collection&gt;</code>&lt;br&gt;<code>&lt;/security-constraint&gt;</code></td>
</tr>
<tr>
<td>WSA administration</td>
<td><code>&lt;security-constraint&gt;</code>&lt;br&gt;<code>&lt;web-resource-collection&gt;</code> &lt;br&gt;<code>&lt;url-pattern&gt;/wsa1/admin/*/&lt;/url-pattern&gt;</code>&lt;br&gt;<code>&lt;auth-constraint&gt;</code>&lt;br&gt;<code>&lt;role-name&gt;WSAdminUser1&lt;/role-name&gt;</code>&lt;br&gt;<code>&lt;role-name&gt;WSAdminUser2&lt;/role-name&gt;</code>&lt;br&gt;<code>&lt;/auth-constraint&gt;</code>&lt;br&gt;<code>&lt;/web-resource-collection&gt;</code>&lt;br&gt;<code>&lt;/security-constraint&gt;</code></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 Java container 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 the following table as Web services.

Table 42: 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>
</tbody>
</table>
This application . . . | Available to . . . | Lets its users . . .
---|---|---
Accounting (Update) | Members of the Accounting department only | Read, modify, and create records in the Accounting system
Human Resources | Members of the Human Resources department only | Read, modify, and create records in the Human Resources system
Work Tracking | All employees | Record the time spent on each project

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 the following table.

**Table 43: 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>acctng</td>
<td>/acctng/</td>
<td>Acctdept</td>
</tr>
<tr>
<td>Accounting-Query</td>
<td>acctngquery</td>
<td>/acctngquery/</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, select the WSA instance in OpenEdge Management or OpenEdge Explorer, and select **Configuration**. Select **Security** to display the Security tab. In the Security tab, 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 Java container's user database, assign each employee the appropriate role, as shown in the following table.
### Table 44: 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 version="1.0" encoding="ISO-8859-1"?>
<web-app
 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
 "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<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>accntgquery_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>accntgquery</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>accntg_servlet</servlet-name>
 <display-name>Web Services Adapter full accountingservlet</display-name>
 <servlet-class>com.progress.wsa.WsaServlet</servlet-class>
</servlet>
</web-app>
<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 constraints here -->
<security-constraint>
   <web-resource-collection>
      <web-resource-name>HR Web Services</web-resource-name>
      <url-pattern>/hr/*</url-pattern>
   </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>
   </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>
   </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>
   </web-resource-collection>
   <http-method>POST</http-method>
</security-constraint>
<web-resource-collection>
  <auth-constraint>
    <role-name>Acctdept</role-name>
  </auth-constraint>
</web-resource-collection>

<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Acme Company Web Services</realm-name>
</login-config>
Using the WSA Management Utility (WSAMAN)

This chapter provides an overview of the functions of the WSAMAN utility.

For details, see the following topics:

• Overview 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 are also accessible in OpenEdge Management or OpenEdge Explorer. For more information, see the online help.

Syntax

Each function of the WSAMAN utility has syntax similar to the following:

Table 46: Syntax

```
wsaman [ -option]... -function
```
[ -option ] ...

One or more options, where each option consists of a name (for example, -appname) and a value (for example, wsa_instance-name).

function

The name of the function (for example, getdefaults).

For more information on the functions of the WSAMAN utility, see Command and Utility Reference on page 357.

Functions for managing a WSA instance

Some functions of the WSAMAN utility allow you to manage a WSA instance. These are summarized in the following table.

Table 47: WSA management functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSAMAN getdefaults on page 400</td>
<td>Displays the default Web service properties associated with a WSA instance</td>
</tr>
<tr>
<td>WSAMAN getprops (WSA) on page 402</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) on page 403</td>
<td>Displays statistics for a WSA instance</td>
</tr>
<tr>
<td>WSAMAN list on page 405</td>
<td>Displays the list of Web service applications that have been deployed to the WSA instance</td>
</tr>
<tr>
<td>WSAMAN query (WSA) on page 406</td>
<td>Displays the following characteristics and properties of a WSA instance:</td>
</tr>
<tr>
<td></td>
<td>• Status (running, not running, or status not known)</td>
</tr>
<tr>
<td></td>
<td>• Whether the Administration function is enabled or disabled</td>
</tr>
<tr>
<td></td>
<td>• Whether the Web Services Description Language (WSDL) function is enabled or disabled</td>
</tr>
<tr>
<td></td>
<td>• Whether the Web service function is enabled or disabled</td>
</tr>
<tr>
<td>WSAMAN resetdefaults on page 407</td>
<td>Resets, to its original values, a WSA instance's default.props file (which contains the default Web service properties associated with the WSA instance)</td>
</tr>
<tr>
<td>WSAMAN resetstats (WSA) on page 408</td>
<td>Resets the statistics for a WSA instance</td>
</tr>
</tbody>
</table>
**Function** | **Description**
---|---
WSAMAN setdefault on page 409 | Sets one of the default Web service properties associated with a WSA instance
WSAMAN setprops (WSA) on page 410 | Temporarily changes the value of one of the properties of a WSA instance; changes last until the Java container is stopped and restarted

For more information on the functions of the WSAMAN utility, see Command and Utility Reference on page 357.

### Functions for managing a Web service

Other functions of the **WSAMAN** utility let you manage a Web service. These are summarized in the following table.

**Table 48: Web service management functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSAMAN deploy on page 396</td>
<td>Deploys a Web service to a WSA instance.</td>
</tr>
<tr>
<td>WSAMAN disable on page 398</td>
<td>Makes a deployed Web service temporarily unavailable to incoming client requests.</td>
</tr>
<tr>
<td>WSAMAN enable on page 398</td>
<td>Makes a deployed Web service available to incoming client requests.</td>
</tr>
<tr>
<td>WSAMAN export on page 399</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>
<tr>
<td>WSAMAN getprops (Service) on page 401</td>
<td>Displays the properties of a Web service. If the Web service is enabled, only the <code>serviceFaultLevel</code> and <code>serviceLoggingLevel</code> 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) on page 402</td>
<td>Displays the statistics for a Web service.</td>
</tr>
<tr>
<td>WSAMAN import on page 403</td>
<td>Deploys a previously exported Web service.</td>
</tr>
</tbody>
</table>
| WSAMAN query (Service) on page 405 | 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) |
### Function and Description

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSAMAN resetprops on page 407</td>
<td>Reinitializes a Web service's <code>friendlyname.props</code> file to the current value of the WSA instance's <code>default.props</code> file.</td>
</tr>
<tr>
<td>WSAMAN resetstats (Service) on page 408</td>
<td>Resets the statistics of a Web service.</td>
</tr>
<tr>
<td>WSAMAN setprops (Service) on page 410</td>
<td>Sets one of the properties of a Web service.</td>
</tr>
<tr>
<td>WSAMAN undeploy on page 411</td>
<td>Undeploys a Web service from a WSA instance.</td>
</tr>
<tr>
<td>WSAMAN update on page 412</td>
<td>Lets you change a Web service's deployment information (stored in the Web Service Mapping (WSM) file) without undeploying and deploying. <strong>Note:</strong> Progress Software Corporation recommends that <code>update</code> never be used on a production system.</td>
</tr>
</tbody>
</table>

For more information on the functions of the WSAMAN utility, see Command and Utility Reference on page 357.
WebSpeed Administration

For details, see the following topics:

- Configuring WebSpeed in Windows
- Configuring WebSpeed on UNIX
- WebSpeed Dynamic Code-page Support
- Connecting WebSpeed to a Data Source
- WebSpeed Security
- Using Active Server Pages with WebSpeed
Configuring WebSpeed in Windows

This chapter explains how to configure WebSpeed to run in Windows.

For details, see the following topics:

- 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.

Configuring the WebSpeed components on Windows

To configure the WebSpeed components:

1. Start the AdminService process on each machine. (See Starting the AdminService on page 158.)

2. Once the AdminService is running, you can use OpenEdge Management or OpenEdge 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 OpenEdge Management or OpenEdge Explorer on page 149.)

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 Overview of the ubroker.properties file on page 157).

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 NSCONFIG on page 372 and WSCONFIG on page 413.)

3. Determine if you must set (or change any preset) WebSpeed environment variables. (See Setting up the WebSpeed environment on page 158.)

4. Use OpenEdge Management or OpenEdge 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 Management or OpenEdge Explorer (or the WTBMAN utility) to start up each WebSpeed Transaction Server instance controlled by that NameServer. (See Starting the WebSpeed Transaction Server and NameServer on page 162.)

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 Setting up WebSpeed on the Web server machine on page 154.)
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 Management or OpenEdge Explorer, the WebSpeed command-line utilities, or the WebSpeed Messenger Administration (WSMAdmin) page in a browser. (See Testing your configuration on page 165 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 Management or OpenEdge 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 > Settings > 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 Management or OpenEdge Explorer

You can use OpenEdge Management or OpenEdge Explorer to manage WebSpeed. Each provides access to the widest array of configuration and operational functions.

OpenEdge Management/OpenEdge Explorer is browser-based and available for management of local and remote servers on any platform. 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.

**Defining or configuring WebSpeed with OpenEdge Management or OpenEdge Explorer**

To define or configure a WebSpeed instance, you can use OpenEdge Management or OpenEdge Explorer.

To define a WebSpeed instance:

1. Make sure the AdminServer is running on the host where you want to configure WebSpeed.
2. Open OpenEdge Management or OpenEdge Explorer.
3. To define a new instance, choose one:
   - From the management console bar, click the **Resources** tab's dropdown arrow. Choose **WebSpeed > New OpenEdge Resource**.
   - Click the **Resources** tab in the management console menu bar. In the Details frame, click **New Resource Monitor**. The **New Resource Monitor** page appears. Click **WebSpeed**.

   The **WebSpeed Configuration** page appears.
4. Type the name of the new WebSpeed broker instance in the field provided.

   **Note**: The WebSpeed broker instance is unique and case-sensitive. It can contain alphanumeric and special characters, except period ("."), colon ("."), or square brackets ('[ ]'). The name must not contain blank spaces.
5. Click **Save**. The **WebSpeed Configuration** page appears, allowing you to configure the WebSpeed broker's properties.

**Modifying an existing WebSpeed configuration**

To modify an existing WebSpeed configuration:

1. Click **Resources** in the management console menu bar. Expand the container name; the main resource types appear in the list frame.
2. In the list frame, expand the **OpenEdge** category.
3. Expand the **WebSpeed** category, and select a WebSpeed Transaction Server Broker. The Details page for the broker appears in the detail frame.
4. Click **Configuration**.
5. Click **Edit**.
6. Select a property category tab and set the properties as required. You can accept the default values, if they are appropriate for your application. Be sure to review and set properties in each category.

   See the OpenEdge Management or OpenEdge 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** — If you want WebSpeed to start whenever you start the AdminServer, select the **Auto start** check box. If you want to use a different working directory than the one specified during the WebSpeed product installation, you can change it here.

The **Operating Mode** is fixed stateless. Specify a nondefault value for the TCP/IP port number where the WebSpeed broker listens for requests.

• **Owner Information** — You can optionally provide **Group name**, **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 Management or 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).

You can also set the TCP/IP version and **jvmargs**; decide whether to enable dynamic property updates; enabling debugging through the broker; provide a broker debugger port number; enable SSL for the broker debugger; use the broker's private key/digital certificate for the debugger; use the debugger's private key/digital certificate alias name; set a password to access the debugger's key/certificate; set a broker debugger passphrase; and set a publish directory.

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**.

You can also set how often the data is flushed by the agent to the broker.

- **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 Management or 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**. You can also specify the maximum size in bytes of the binary files that WebSpeed handles.

  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. Also, set properties for the maximum time in seconds that a remote procedure can execute on a given Web server.

  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 - Security and Auditing*. 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; and 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.

7. Choose the OK button to save the configuration.

For information about how to use OpenEdge Management or OpenEdge Explorer to configure WebSpeed, see the 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, perform an App Server Refresh, and shut down the Transaction Server.

For more information about the WTBMAN utility, see WTBMAN on page 415.

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 WSCONFIG on page 413.

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 NSMAN on page 374.
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 NSCONFIG on page 372.

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 the following table.

Table 49: 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>WSASP⁴</td>
<td>wsasp.dll</td>
</tr>
<tr>
<td>Netscape</td>
<td>NSAPI</td>
<td>wnsa.dll</td>
</tr>
<tr>
<td>CGI-compatible</td>
<td>CGI</td>
<td>cgiip.exe</td>
</tr>
</tbody>
</table>

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

⁴ The WSASP Messenger calls WebSpeed applications from an Active Server Page. It cannot coexist with the ISAPI Messenger.
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 download a compatible Messenger. Access the Progress Software Download Center and log in to your account at [http://www.progress.com/esd](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`).

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

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

---

**OpenEdge Application Server: Administration**

155
Table 50: Netscape Web server configuration

<table>
<thead>
<tr>
<th>New Line</th>
<th>Description</th>
</tr>
</thead>
</table>
| Init fn=load-modules  
shlib="pathname"  
funcs=WSNSAinit,WSNSAdefault,WSNSAshutdown,WSNSAWebSpeedCheck | 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. |
| Init fn=WSNSAinit | 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. |
| NameTrans fn=WSNSAWebSpeedCheck | This NameTrans line must appear before any existing NameTrans commands. You must add it within the block of commands delimited by <Object name=default> and </Object>. This line informs the Web server to handle WebSpeed requests properly. |
| Service method=(GET|POST|HEAD)  
fn=WSNSAdefault | This Service line must appear before any existing Service commands. You must add it within the block of commands delimited by <Object name=default> and </Object>. This line informs the Web server to handle WebSpeed requests properly. |

Each line you add to obj.conf must be on a single line. Do not add line breaks within a command line. Use forward slashes (/) in pathnames. Here is an excerpt from a sample obj.conf file (the additions that you must make for the WebSpeed Messenger are bold):

```
Init ...  
Init ...  
# The following directive is a single line; it contains no line breaks  
Init fn=load-modules shlib="c:/Program Files/OpenEdge/bin/wsnsa.dll"  
   funcs=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 Management/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 Management or 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 Overview of the `ubroker.properties` file on page 179 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 > Settings > 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 by using OpenEdge Management/OpenEdge Explorer or the WebSpeed command-line utilities.

### Setting environment variables

Environment variables required by WebSpeed are set during installation in the `ubroker.properties` file. You might need to set the `PROPATH` and other standard environment variables (for example, `DLC`) on the WebSpeed Transaction Server machine.

**Note:** In distributed configurations, you must edit the appropriate environment variables on each machine where you have WebSpeed components installed.

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

When you install the WebSpeed Transaction Server, the installation process sets the **PROPATH** for you in the `ubroker.properties` file. **PROPATH** initially includes a number of subdirectories in your installation directory. In addition, the **PROPATH** includes a dot (.) directory reference. When the agent sees the dot, the process substitutes the name of its current working directory. For example, the agents resolve the dot to their broker's default directory, which is the working directory.

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

**Working directory settings**

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

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

**Configuring WebSpeed and NameServer log files**

For disk management reasons, you might want to specify a 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 Management or OpenEdge Explorer or by directly editing the `ubroker.properties` file. For more information, see *Configuring WebSpeed components* on page 159.

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 *Maintaining the WebSpeed Transaction Server and NameServer log files* on page 164.

**Configuring WebSpeed components**

You can use OpenEdge Management or OpenEdge Explorer to create and configure the NameServer and the WebSpeed broker. You can use OpenEdge Management or OpenEdge Explorer or the command-line utilities to manage the configuration.

For more information about creating, configuring, and managing with OpenEdge Management or OpenEdge Explorer, see the OpenEdge Management or OpenEdge Explorer online help.

For information about managing with the command-line utilities, see *WebSpeed command-line utilities* on page 153.
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.

Eliminating NameServer using OpenEdge Management or OpenEdge Explorer

To eliminate the NameServer using OpenEdge Management or OpenEdge Explorer:

1. Select your broker and click Configuration.
2. Click Edit.
3. On the Broker tab, 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.

Eliminating NameServer using the ubroker properties file

You can also eliminate the NameServer by directly editing the ubroker.properties file, although using OpenEdge Management or 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:

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

5. Add and set the following property:

```java
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`:

```java
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 Management/OpenEdge Explorer or by editing the `priorityWeight` property in the `ubroker.properties` file.

**Percentage weight factors**

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

**Table 51: Weight factors based on percentage**

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

**Arbitrary weight factors**

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

**Table 52: Weight factors based on arbitrary sums**

<table>
<thead>
<tr>
<th>WebSpeed Transaction Server name</th>
<th>Weight factor</th>
<th>% of time selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS1</td>
<td>2</td>
<td>2/7</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 *Starting the AdminService* on page 158).

2. Start an existing NameServer or create a new NameServer instance. You can create and start a NameServer by using OpenEdge Management/OpenEdge Explorer, or you can edit the `ubroker.properties` file to create an instance and then use the `NSMAN` utility to start the instance. When you configure a NameServer instance, you can set it to start up by default whenever the AdminService starts.
Note: The NameServer can be on any machine in your network, even on UNIX.

If you are using OpenEdge Management/OpenEdge Explorer, see the online help for information about creating and starting an instance. If you are editing the ubroker.properties file, see Editing the ubroker.properties file on page 179.

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

See the OpenEdge Management/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:

```
wbtman -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:

```
wbtman -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 Management/OpenEdge Explorer prompts you for a user name (if you do not supply it) and password.
By using either OpenEdge Management/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 Management or OpenEdge Explorer online help, Using OpenEdge Management or OpenEdge Explorer to check status on page 165, and Managing the WebSpeed Transaction Server on page 166.

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 KEEPMESSAGES` 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 Configuring WebSpeed and NameServer log files on page 159.

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 Starting the WebSpeed Transaction Server and NameServer on page 162.
Testing your configuration

To test your configuration:

- Use OpenEdge Management or 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 (WSMAadmin)
- Run the status.p procedure from a Web browser

Using OpenEdge Management or OpenEdge Explorer to check status

You can check the WebSpeed configuration status from OpenEdge Management or OpenEdge Explorer. Select the WebSpeed broker in the list frame on the left, and click Status in the detail frame on the right. See the OpenEdge Management or 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 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 [*Managing the WebSpeed Messenger*](#) on page 168 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

• Perform an AppServer Refresh

• Stop the Transaction Server

• Display command-line help
Dynamically starting additional agents

To start additional agents, enter the following command:

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

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

Trimming running agents

To trim agents, enter the following command:

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

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

AppServer Refresh

The Application Server (AppServer) Refresh enhancement allows the users to trim and restart individual agents gradually to allow new agents to replace older agents. Users can issue a `-refresh` command form ASBMAN or WTBMAN to perform the refresh action. The refresh operation tries to maintain run-time capacity at equal to or greater than what it was at the time refresh started. AppServer Refresh can help perform the following actions:

- Reload r-code
- Trim file system allocation
- Recover process memory

**Note:** The refresh action may take considerable time to finish and if you are using the App Server Refresh feature to load new r-code, you should be aware that the clients that were connected to the App Server, before the refresh action was performed, will have two versions of the r-code between the CONNECT and DISCONNECT calls. All the clients that connect after the refresh is performed will get new agents and will run new r-code.

For information on how to issue a refresh command and other details, refer to ASBMAN on page 362 and WTBMAN on page 415.

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=appservicename]?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?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.
Configuring WebSpeed on UNIX

This chapter explains how to configure WebSpeed to run on UNIX.

For details, see the following topics:

- 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.

Configuring the WebSpeed components on UNIX

To configure the WebSpeed components:

1. Start the AdminServer process on each machine by using the PROADSV utility. (See Starting the AdminServer on page 181.)

2. Once the AdminServer is running, you can then use OpenEdge Management/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 also use OpenEdge Management/OpenEdge Explorer, which runs in a Web browser, 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 Overview of the ubroker.properties file on page 179).

   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 NSCONFIG on page 372 and WSCONFIG on page 413).

3. Determine if you need to set (or change any preset) WebSpeed environment variables (see Setting up the WebSpeed environment on page 181).

4. Use the NSMAN utility (or OpenEdge Management/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 Management/OpenEdge Explorer remotely) to start up each WebSpeed Transaction Server instance controlled by that NameServer (see Starting the WebSpeed Transaction Server and NameServer on page 183).

   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 Setting up WebSpeed on the Web server machine on page 173).
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 Management/OpenEdge Explorer, or the WebSpeed Messenger Administration (WSMAdmin) page in a browser (see Testing your configuration on page 185 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 Management or 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 Management or OpenEdge Explorer

You can use OpenEdge Management or OpenEdge Explorer to manage WebSpeed. Each provides access to the widest array of configuration and operational functions.

OpenEdge Management/OpenEdge Explorer is browser-based tool available for management of local and remote servers on any platform. 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 on page 104.

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 WSACONFIG on page 395.

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 NSMAN on page 374.

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 NSCONFIG on page 372.
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 the following table.

Table 53: UNIX WebSpeed Messengers

<table>
<thead>
<tr>
<th>Web server type</th>
<th>Messenger executable</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSAPI (Netscape)</td>
<td>wnsa.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 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 \( \text{install-dir/https-host-name/config/obj.conf} \). (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. The following table describes the changes you must make to obj.conf.

Table 54: Netscape Web server configuration

<table>
<thead>
<tr>
<th>New line</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Init fn=load-modules**  
  shlib="pathname"  
  funcs=WSNSAinit,WSNSAdefault,  
  WSNSAshutdown,WSNSAwebspeedCheck | This pathname is the absolute path to the Messenger shared object, wsnsa.dll. 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. |
| **Init fn=WSNSAinit** | 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. |
| **NameTrans fn=WSNSAwebspeedCheck** | This NameTrans line must appear before any existing NameTrans commands. You must add it within the block of commands delimited by \(<\text{Object name=default}>\) and \(<</\text{Object}>\). This line informs the Web server to handle WebSpeed requests properly. |
| **Service method=(GET|POST|HEAD)**  
  fn=WSNSAdefault | This Service line must appear before any existing Service commands. You must add it within the block of commands delimited by \(<\text{Object name=default}>\) and \(<</\text{Object}>\). This line informs the Web server to handle WebSpeed requests properly. |
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="/usr/dlc/bin/wsnsa.dll"
    funcs=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>...
```

**Editing the Netscape Web server start file**

The Netscape Web server must be able to locate the WebSpeed Messenger and the resources that the Messenger requires for configuration information and messages. Edit the Netscape Web server's start file (for Netscape Web servers, this file is `install-dir/https-hostname/start`) to set the environment variables described in the following table.

**Table 55: NSAPI Messenger settings**

<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: <code>/install-path/prommsgs</code></td>
</tr>
<tr>
<td>PROCFG</td>
<td>The pathname of the configuration file: <code>/install-path/progress.cfg</code></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>
</tbody>
</table>
The following is a sample start file for AIX that includes the necessary WebSpeed information:

```bash
#!/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`.

---

5 On AIX only, you need to add `/usr/lib` to the shared library search path environment variable.
The following is an edited version of `install-path/bin/wspd_cgi.sh`:

```
# 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 -i wsbroker1
# 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 `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 `-f` option to specify a named configuration file. The following table describes the various options.

**Table 56: Options for invoking the CGI Messenger**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cgiip</code></td>
<td>The Messenger uses the Transaction Server that you defined as the Default Service in the <code>ubroker.properties</code> file.</td>
</tr>
<tr>
<td><code>cgiip host-name port-num</code></td>
<td>The Messenger uses the Transaction Server whose host name and port number you specify.</td>
</tr>
<tr>
<td><code>cgiip -i WS-service</code></td>
<td>The Messenger uses the Transaction Server service whose name you specify.</td>
</tr>
<tr>
<td><code>cgiip -f cnf-file</code></td>
<td>The Messenger uses the configuration file that you specify. This option overrides the <code>ubroker.properties</code> 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:

```bash
# 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`.

[178] OpenEdge Application Server: Administration
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 Management/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.

You can also use OpenEdge Management or OpenEdge Explorer to create and configure instances of the WebSpeed Transaction Server or the NameServer on UNIX.

If you edit the configuration by hand, without OpenEdge Management or 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 Management or 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 is unique and case-sensitive. It can contain alphanumeric and special characters, except period ("."), colon (":"), or square brackets ("["]"). The name must not contain blank spaces.

• 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:


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 Command and Utility Reference on page 357.

**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:

```bash
proadsv -start
```

For more information on using the `PROADSV` utility, see `PROADSV` on page 376.

**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 Management or 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 Management or 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 Management/OpenEdge Explorer or by directly editing the `ubroker.properties` file. For more information, see Configuring WebSpeed components on page 182.

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 Maintaining the WebSpeed Transaction Server and NameServer log files on page 184.

Configuring WebSpeed components

You can use OpenEdge Management or 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 Management or OpenEdge Explorer, see the OpenEdge Management or OpenEdge Explorer online help.

For information about managing with the command-line utilities, see WebSpeed command-line utilities on page 171.

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 Management or OpenEdge Explorer or by editing the `priorityWeight` property in the `ubroker.properties` file.
For more information on load balancing and fault tolerance, see Understanding the NameServer's load-balancing option on page 161, 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 Eliminating the NameServer on page 160.

---

**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 Starting the AdminServer on page 181.)

2. Start an existing NameServer or create a new NameServer instance. You can create and start a NameServer by using OpenEdge Management or 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 Management or OpenEdge Explorer, see the online help for information about creating and starting an instance. If you are editing the ubroker.properties file, see Editing the ubroker.properties file on page 179.

   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 Management or 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 Management or OpenEdge Explorer, see the online help for information about creating and starting an instance. If you are editing the ubroker.properties file, see Editing the ubroker.properties file on page 179.
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 Management/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 Management or OpenEdge Explorer online help. Also see Using OpenEdge Management or OpenEdge Explorer to check status on page 185 and Managing the WebSpeed Transaction Server on page 187.

### 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 Configuring WebSpeed and NameServer log files on page 182.

### 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 Management or OpenEdge Explorer to start the NameServer or the WebSpeed Transaction Server. See the OpenEdge Management or OpenEdge Explorer online help for information.

Testing your configuration

To test your configuration:

- Use OpenEdge Management or 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 (WSMAadmin)
- Run the `status.p` procedure from a Web browser

Using OpenEdge Management or OpenEdge Explorer to check status

You can check the WebSpeed configuration status from OpenEdge Management or OpenEdge Explorer. Select the WebSpeed broker in the list frame on the left, and click Status in the detail frame on the right.

See the OpenEdge Management or 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:

```bash
proadsv -query
```

To query a local NameServer, use the NSMAN utility, which is installed in `install-path/bin`, as shown:

```bash
nsman -name NameServer-name -query
```

To query a remote NameServer, add the following parameters to the command:

```bash
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:

```bash
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:

```bash
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 [Managing the WebSpeed Messenger](#) on page 188 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 Management or OpenEdge Explorer to manage and validate the Transaction Server. (With OpenEdge Management/OpenEdge Explorer, you can also configure the Transaction Server). See the OpenEdge Management or 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:

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

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

```plaintext
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.
WebSpeed Dynamic Code-page Support

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.

For details, see the following topics:

• Introduction
• Basic browser-agent interaction
• How dynamic code-page support works
• Summary of code-page conversions
• Configuring the agent
• Tasks for the WebSpeed developer
• Storing the MIME code-page name

Introduction

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 **META** tag with the MIME version of the code-page name, as in the following:

```html
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset="windows-1250">
```

• If the Web page is created using dynamic HTML, the application uses the **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 **Connecting WebSpeed to a Data Source** on page 197.

**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 \texttt{CPINTERNAL} code page. To simplify meeting this requirement, make the agent's \texttt{CPINTERNAL} code page \texttt{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.

\textbf{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 \texttt{<META>} tag's \texttt{HTTP-EQUIV} attribute. See the first code example in \textit{Storing the MIME code-page name} on page 194.

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 \texttt{CPINTERNAL} code page. Similarly, just before the agent sends a Web page back to the browser, it automatically converts it from the agent's \texttt{CPINTERNAL} code page to the browser's code page.

\section*{Determining the MIME code-page name}

The application must store the MIME code-page name, not the OpenEdge code-page name. The following table gives the MIME name and the OpenEdge name for several common code pages.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{MIME name} & \textbf{OpenEdge name} \\
\hline
iso-8859-2 & iso8859-2 \\
\hline
Shift\_JIS & SHIFT-JIS \\
\hline
TIS-620 & 620-2533 \\
\hline
windows-1250 & 1250 \\
\hline
\end{tabular}
\caption{MIME and OpenEdge code-page equivalents}
\end{table}

\textbf{Note:} The above table 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 \texttt{adecomm/convcp.p}.

For more information on converting between MIME and OpenEdge code-page names, see \textit{Converting code-page names between MIME and OpenEdge formats} on page 195.

\section*{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 \texttt{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>
<BODY>
<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">
<INPUT TYPE="submit" VALUE="Submit">
</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:

```html
```

- 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:
Table 58: Syntax

```
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`:

```
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`:

```
RUN adecomm/convcp.p(cProCharset, "toMime":U, OUTPUT cMimeCharset).
```

**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 Creating an international Web site on page 201 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.

For details, see the following topics:

- 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 Management or 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 qualify 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 59: Remote connection parameters on page 199) 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 the following table for a remote connection.

Table 59: 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 \etc\services 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:

```bash
srvrStartupParam=-p web\objects\web-disp.p -cpstream iso8859-1 -weblogerror -db sports2000 -H pluto -S mysvl2
```

**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, `mysvl2`.

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 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.

The following figure 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 6: The local 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 OpenEdgeDataServer 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 Server startup parameters field on the Server tab in OpenEdge Management or 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 recommend 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 WebSpeed Dynamic Code-page Support on page 191.

The following figure shows a WebSpeed configuration that supports an international Web site.

**Figure 8: WebSpeed configuration that supports an international Web site**

![WebSpeed configuration diagram]

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.

You can use OpenEdge Management or OpenEdge Explorer to configure the agents. You add the Parameter File (-pf) startup parameter and name of your parameter file to the Server startup parameters field for the agent.

On UNIX, use OpenEdge Management/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:

```
-db intldb -cpinternal SHIFT-JIS -cpstream SHIFT-JIS -d ymd
```
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:

```bash
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.

**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 8: WebSpeed configuration that supports an international Web site on page 202 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.

For details, see the following topics:

- 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** on page 208
- **Disable the WebSpeed Messenger Administration utility** on page 209
- **Establish the WebSpeed Messenger Administration Internet Protocol List (Optional)** on page 210
- **Check the status of the Debug mode** on page 211
You can access these properties from either OpenEdge Management/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.

The following table details the differences between production and development modes.

### Table 60: Agent application mode descriptions

<table>
<thead>
<tr>
<th>When the agent application mode is set to . . .</th>
<th>Then this property . . .</th>
</tr>
</thead>
</table>
| **Production** | • Prohibits Web objects specified in the URL with the path install-path\src\web\ from being run  
• Disables these utility commands: debug, reset, and ping  
• Disables access to WebTools  
• 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  
• Ignores debugging via the URL, ?debug=xxx, even if the Debug mode is enabled |
| **Development (default)** | • Allows unlimited access to run Web objects specified in the URL with the path install-path\src\web\  
• Enables these utility commands: debug, reset, and ping  
• Enables access to WebTools  
• Allows users to access specific development files  
• Enables debugging via the URL, ?debug=xxx, if Debug mode is enabled |

You can use OpenEdge Management or OpenEdge Explorer to change the value of the Agent application mode from development to production.

Another method of changing the application mode from development to production is to edit the ubroker.properties file.

**Changing the application mode with OpenEdge Management or OpenEdge Explorer**

To change the application mode with OpenEdge Management or OpenEdge Explorer:

1. Click **Resources** in the management console menu bar. Expand the container name; the main resource types appear in the list frame.
2. In the list frame, expand the **OpenEdge** category.
3. Expand the **WebSpeed** category, and select a WebSpeed broker. The **Details** page for that broker appears in the detail frame.

4. In the **Command and control** section of the page, click **Configuration**. The **WebSpeed Configuration** page opens with the **Broker** properties in focus.

5. Click the **Agent** tab.

6. Click **Edit**.

7. Under the **General** category, set the **Server application mode** property.

### Changing the application mode using 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 WSMAdmin with the WebSpeed Messenger Administration Internet Protocol List option, you can leave the WebSpeed Messenger Administration utility enabled in Production mode. See *Establish the WebSpeed Messenger Administration Internet Protocol List (Optional)* on page 210 for more information.

### Disabling WSMAdmin

You can use OpenEdge Management or OpenEdge Explorer to disable the WSMAdmin utility.

To disable the WSMAdmin utility with OpenEdge Management or OpenEdge Explorer:
1. Click **Resources** in the management console menu bar. Expand the container name; the main resource types appear in the list frame.

2. In the list frame, expand the **OpenEdge** category.

3. Expand the **Messengers** category, and select the specific messenger whose internal commands you want to disable. The **Details** page for that messenger appears in the detail frame.

4. Click **Configuration**.

5. Click **Edit**.

6. On the **Advanced** tab, deselect the check box next to the **Internal administration command**.

7. 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.

**Disabling the WSMAdmin utility using the ubroker.properties file**

To 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.

You can add IP addresses using OpenEdge Management or OpenEdge Explorer. Alternatively, you can define specific IP addresses using the **ubroker.properties** file.

**Adding IP addresses using OpenEdge Management or OpenEdge Explorer**

To add IP addresses using OpenEdge Management or OpenEdge Explorer:
1. Click **Resources** in the management console menu bar. Expand the container name; the main resource types appear in the list frame.

2. In the list frame, expand the **OpenEdge** category.

3. Expand the **Messengers** category, and select the specific messenger whose internal commands you want to enable. The **Details** page for that messenger appears in the detail frame.

4. Click **Configuration**.

5. Click **Edit**.

6. On the Advanced tab, check **Internal administration command**.

7. In the **Authorized IP list** field, type one or more unique IP addresses.

8. Click the **Save** button.

**Defining specific IP addresses using the ubroker.properties file**

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.

You can check the status of the Debug mode using OpenEdge Management or OpenEdge Explorer. Alternatively, you can check the status using the **ubroker.properties** file.

**Checking the status using OpenEdge Management or OpenEdge Explorer**

To check the status of the Debug mode using OpenEdge Management or OpenEdge Explorer:

1. Click **Resources** in the management console menu bar. Expand the container name; the main resource types appear in the list frame.

2. In the list frame, expand the **OpenEdge** category.
3. Expand the **WebSpeed** category, and select the broker. The **Details** page for that broker appears in the detail frame.

4. Click **Configuration**.

5. Click **Edit**.

6. Select the **Agent** tab. Under Advanced Features, locate the **Debug mode** field, which has the following three options: **Enabled**, **Disabled**, or **Default**.

   The server application mode value governs which **Debug mode** values are available. For example:

   - If the value set for the 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 application mode is **Development**, you can set values for the Debug mode that allow you to enable or disable debugging.

### Checking the status using ubroker.properties file

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. The following table lists some key default port numbers to consider changing.

<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 identify 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 a more information about security issues as they pertain to WebSpeed configurations and their integration with firewalls, see Maximizing WebSpeed compatibility with your firewall on page 217.

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 - Security and Auditing* 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 Management/ 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 client connections box in the SSL properties category in OpenEdge Management or 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 Management or OpenEdge Explorer, or manually edit the `ubroker.properties` file.

- 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 Management /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 Management or 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 Management or 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 Management or 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 Management or 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** on page 217
- **Important firewall considerations** on page 218
- **Network Address Translation (NAT) and the WebSpeed distributed configuration** on page 218
- **Using the NameServer client port range value settings with a firewall** on page 220

**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.
The following table identifies the network communications requirements for your WebSpeed configuration.

**Table 62: 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 in either OpenEdge Management/OpenEdge Explorer or 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 in either in OpenEdge Management/OpenEdge Explorer or 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.

You can make a client connection through a NAT firewall using OpenEdge Management or OpenEdge Explorer. Alternatively, you can make a client connection through a NAT firewall using `ubroker.properties` file.

### Making a client connection using OpenEdge Management or OpenEdge Explorer

To make a client connection through a NAT firewall using OpenEdge Management or OpenEdge Explorer:

1. Click **Resources** in the management console menu bar. Expand the container name; the main resource types appear in the list frame.
2. In the list frame, expand the **OpenEdge** category.
3. Expand the **WebSpeed** category, and select a broker. The **Details** page for that broker appears in the detail frame.
4. Click **Configuration**.
5. Click **Edit**.
6. Select the **Broker** tab to see the **Controlling NameServer** Information.
7. Select the Registration Mode value you want to set. The following table defines each value.

<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>

8. Click **OK**.

### Making a client connection using `ubroker.properties` file

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. The following table defines the values from which you can choose to enter in the `registrationMode` property.

Table 64: RegistrationMode Connection—ubroker.properties file

<table>
<thead>
<tr>
<th>Enter this Registration Mode . . .</th>
<th>To Identify . . .</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-HostName</td>
<td>A specific a hostname or IP address. You must:</td>
</tr>
<tr>
<td></td>
<td>• Enter the value <code>Register-HostName</code> in the <code>registrationMode</code> property field.</td>
</tr>
<tr>
<td></td>
<td>• Enter the hostname, or the IP address of the hostname, in the <code>hostName</code> property field to identify where the WebSpeed broker is running.</td>
</tr>
<tr>
<td></td>
<td>• Enter a description of the type of IP address you entered in the <code>description</code> property field.</td>
</tr>
<tr>
<td></td>
<td>The WebSpeed broker passes either the hostname or the IP address value defined in the <code>hostName</code> property to the NameServer when the broker registers. Clients connect to the host using either the <code>hostName</code> or the IP address defined.</td>
</tr>
</tbody>
</table>

*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 \texttt{minNSClientPort} and a client port range maximum with \texttt{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 \texttt{minNSClientPort} also must be less than \texttt{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.
Using Active Server Pages with WebSpeed

This chapter contains information for developers who want to use Microsoft's Active Server Page (ASP) technology in conjunction with their WebSpeed applications.

For details, see the following topics:

- 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 the following figure.

**Figure 9: The WebSpeed ASP Web Page**
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 Management or OpenEdge Explorer by selecting Application service names on `wsbroker1`. Make sure that the Supports default service check box is checked.
Messaging and ESB Administration

For details, see the following topics:

• OpenEdge Adapter for SonicMQ Administration
• Configuring and Managing the OpenEdge Adapter for Sonic ESB
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.

For details, see the following topics:

- 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.
Chapter 15: OpenEdge Adapter for SonicMQ Administration

**Note:** Progress OpenEdge also provides a generic Java Message Service (JMS) adapter for messaging. This generic JMS adapter can operate with any JMS-compliant vendor, for example, SonicMQ, IBM WebSphereMQ, and ActiveMQ. For more information on this adapter, see *OpenEdge Development: Messaging and ESB*.

---

**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 the Configuring and Managing the AppServer Internet Adapter on page 75 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 SSL-enabled BrokerConnect operation on page 230 for more information.

If you are working with the generic Java Message Service (JMS) adapter for messaging, it supports only the BrokerConnect mode. For more information on this adapter, see *OpenEdge Development: Messaging and ESB*.

**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 Management/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 - Security and Auditing.

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 Configuring BrokerConnect on page 233. For more information on SSL support in OpenEdge, see OpenEdge Getting Started: Core Business Services - Security and Auditing. For more information on configuring the AIA for Internet access to OpenEdge server sessions, see Configuring an AIA with OpenEdge Management or OpenEdge Explorer on page 80.

Configuring and administering the OpenEdge Adapter for SonicMQ

The following sections describe:

- Configuring ClientConnect on page 232
- Configuring ServerConnect on page 232
- Configuring BrokerConnect on page 233

Note: Progress OpenEdge also provides a generic Java Message Service (JMS) adapter for messaging. This generic JMS adapter can operate with any JMS-compliant vendor, for example, SonicMQ, IBM WebSphereMQ, and ActiveMQ. For more information on this adapter, see OpenEdge Development: Messaging and ESB.
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:

**Table 65: Using SMQConnect on a client example**

```plaintext
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 Management or OpenEdge Explorer.

To enable the AppServer or WebSpeed server for ServerConnect from OpenEdge Management or OpenEdge Explorer:

1. Click **Resources** in the management console menu bar. Expand the container name; the main resource types appear in the list frame.
2. In the list frame, expand the **OpenEdge** category.
3. Expand the **AppServer** or **WebSpeed** category, and select a broker. The **Details** page for that broker appears in the detail frame.
4. Click **Configuration**.
5. Click **Edit**.
6. Click the **Messaging** tab, and select the **SonicMQ ServerConnect enabled** check box.
7. On the **Agent** tab and the **Broker** tab, select unique broker and server log filenames.
8. On the **Agent** tab, 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 Management or OpenEdge Explorer.

To configure ServerConnect logging for an AppServer, see Configuring and Managing the AppServer on page 39. To configure ServerConnect logging for a WebSpeed Transaction Server, see Configuring WebSpeed in Windows on page 147.

### Configuring BrokerConnect

BrokerConnect is a Unified Broker product and part of the Progress Unified Broker framework. You can use OpenEdge Management or OpenEdge Explorer to start, stop, get status, add, delete, and edit properties of BrokerConnect.

To define or 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 Management or OpenEdge Explorer from the Progress OpenEdge program group.
3. To define a new adapter instance or configure an existing instance, do the following:
   - To define a new instance, click the Resources tab's dropdown arrow from the management console bar. Choose New OpenEdge Resource > SonicMQ Adapter. Click SonicMQ Adapter. Type the name of the new broker instance in the field provide, and click Save.
   - To configure an existing instance, click the Resources tab in the management console menu bar. Expand the container name, and then OpenEdge, and then SonicMQ Adapter. Select the instance you want to configure; then click Configuration in the Details frame.

   **Note:** The SonicMQ broker instance is unique and case-sensitive. It can contain alphanumeric and special characters, except period ("."), colon (":"), or square brackets ("[]"). The name must not contain blank spaces.

4. Click Edit, 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 Setting the CLASSPATH on page 239.

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 Group name, 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. Also, choose the broker logging entry types (a single entry or a comma-delimited list), the broker log file entry types, the broker log file threshold size, and the maximum number of broker log files.

- **Advanced Features** — Specify the maximum number of client connections (Maximum client instances) that the BrokerConnect can support at one time. Also, you can 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. In addition, you can set the load balancing priority weight; choose the TCP/IP version; decide whether you want to enable dynamic property updates; determine how the AppServer responds to a connection issues.

5. 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.

6. 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 - Security and Auditing.

7. 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.

8. 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 Management or 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 Management or 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 the following table.

**Table 66: 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 <code>ubroker.properties</code> 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>

  **Note:** The `product-instance-name` is unique and case-sensitive. It can contain alphanumeric and special characters, except period ("."), colon (";"), or square brackets ("[]"). The name must not contain blank spaces.
Server startup parameters

You can optionally specify the attributes described in the following table in the `srvrStartupParam` property of BrokerConnect.

### Table 67: 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)</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:
Table 68: Syntax

```
adaptconfig [  
    [ [-name adapter-broker]] [-propfile path-to-properties-file]]  
    [ -validate ] | -help 
```

For information on using the `adaptconfig` utility, see `ADAPTCONFIG` on page 359.

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:

Table 69: 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 `ADAPTMAN` on page 360.

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.

**Note:** If you want to set the classpath for using the generic Java Message Service (JMS) adapter, see *OpenEdge Development: Messaging and ESB*.

### 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 (`-cp internal 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.

For details, see the following topics:

• 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
• Starting the ESB container
• 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
  • Progress Developer Studio for OpenEdge
  • 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
  • Progress Developer Studio for OpenEdge
  • 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:
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 **OpenEdge Web Services**, as shown in Figure 10: OpenEdge Native Services on page 244 and Figure 11: OpenEdge Web Services on page 244.

**Figure 10: OpenEdge Native Services**

![OpenEdge Native Services](image)

**Figure 11: OpenEdge Web Services**

![OpenEdge Web Services](image)
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 on page 246

Creating an OpenEdge service instance on page 248

Editing an instance of an OpenEdge service on page 253

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 Progress Developer Studio for OpenEdge 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 Management or 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** on page 246

• **Generating a WSDL file from the OpenEdge Resource Editor** on page 251

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 *Creating an OpenEdge service instance* on page 248). 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 Reference to OpenEdge Web Service Properties on page 343.

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 Adding the WSM or WSD resource to the Sonic ESB Directory Service on page 246.

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. The first six fields, in **Service Maintenance**, specify the properties common to all Sonic ESB services. Only the **Service Name** is required.

5. 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 Entry 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 **Generating a WSDL file from the OpenEdge Resource Editor** on page 251.

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:

      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 Adding the WSM or WSD resource to the Sonic ESB Directory Service on page 246.
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 from 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 [Reference to OpenEdge Web Service Properties on page 343](#).

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.

To generate a WSDL file from the Resource Editor:

1. Open the Sonic Management Console.

2. Click the **Configure** tab.

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 <code>local</code> 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.

**Associating the WSDL file with the service**

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:

![Import Screenshot](image)

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 Creating an OpenEdge service instance on page 248.

**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:

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 window]

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.
Starting the ESB 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 Editing an instance of an OpenEdge service on page 253) or as the default for services deployed to a given adapter instance (see Editing the default service properties on page 247). 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 Management/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 SSL-enabled AppServer operation on page 45 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 - Security and Auditing.*

**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 *Reference to OpenEdge Web Service Properties* on page 343.
For details, see the following topics:

- REST Management Agent Administration
- REST Web Application Administration
- Using the RESTMAN Utility
- REST Management Agent and REST Web Application Security Configurations
A REST Management Agent is a component of Progress OpenEdge that you use to deploy, configure, and manage a Java container's (or Web server's) OpenEdge REST Web applications. Technically, the REST Management Agent is a Java Web application that acts as an intermediary for you to communicate with the deployed REST Web applications. It helps you enable or disable the application, collect statistics, and provide a run-time configuration for a REST Web application.

OpenEdge provides three clients—OpenEdge Management, OpenEdge Explorer, and RESTMAN utility—that use a REST Management Agent to manage the REST Web applications.

For more information on working with OpenEdge Management and OpenEdge Explorer to administer REST Management Agent and the REST Web applications, see the OpenEdge Management and OpenEdge Explorer: Configuration manual. For more information on REST Web application administration and on using RESTMAN utility, see Using the RESTMAN Utility on page 303.

This chapter describes the administrative tasks for REST Management Agent.

For details, see the following topics:

- Installing and Configuring REST Management Agent
- Managing REST Management Agent
- Security Configurations for a REST Management Agent

Installing and Configuring REST Management Agent

After installing OpenEdge, you might have to perform one or more post-installation configuration tasks in order to deploy REST Web applications.

These tasks are described in the following sections:
Installing a REST Management Agent into a Java container

As part of the OpenEdge installation, REST Management Agent is installed in the OpenEdge installation directory, OpenEdge-install-dir\servlets\rest.

Note:
- If you are using the Tomcat Web server installed along with the Progress Developer Studio for OpenEdge, you need not perform the following step. Instead, you can administer the Tomcat Web server using the protc utility as opposed to using the catalina.sh or catalina.bat scripts provided by Apache Tomcat. For more information on the protc utility, see Customizing Apache Tomcat Environment using the protc utility on page 263.
- If you are not using the Tomcat Web server installed along with the Progress Developer Studio for OpenEdge, it is recommended that you use Apache Tomcat version 7.0.30 or later to deploy and run OpenEdge REST Web applications. Additionally, running OpenEdge REST Web applications requires Java 1.6 or later and Servlet API standard 3.0.

To start working with REST Web applications, you must configure your Web server to recognize the REST Management Agent. To configure your Web server, copy the REST Management Agent, oerm.war, from the directory, OpenEdge-install-dir\servlets\rest, to the Web server deployment directory.

For instance, if you are using a Tomcat Web server, you copy oerm.war to the $CATALINA_HOME/webapps directory.

Note: If you are using Tomcat Web server, you can also deploy oerm.war using the Tomcat manager. For more information, see the Tomcat manager documentation.

The following table shows how to perform the copy and paste operation using the command prompt in your operating system:

Table 70: Moving the installed sample Web application

<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 .\servlets\rest\oerm.war destination-directory</td>
</tr>
<tr>
<td>UNIX</td>
<td>cp -r -p .\servlets\rest\oerm.war destination-directory</td>
</tr>
</tbody>
</table>

Note: If the Web server is already running, you must restart the Web server for the new configuration settings to take effect.
You can query the REST Management Agent by browsing to the URL http://localhost:8980/oerm/index.html to check whether the REST Management Agent has been correctly installed and configured. For instance, if you are using the Tomcat server and you are using the default ports, you can browse the URL http://localhost:8980/oerm/index.html.

OpenEdge Management or OpenEdge Explorer allows you to install REST Management Agent instances. For more information, see the OpenEdge Management and OpenEdge Explorer: Configuration.

Customizing Apache Tomcat Environment using the protc utility

The protc utility allows you to administer the Tomcat Web server that is installed along with Progress Developer Studio for OpenEdge. The protc utility allows you to execute supported Tomcat Catalina commands after applying startup and shutdown preferences defined by environment variables that you set in a proset.env (UNIX) or proset.bat (Windows). You must set these environment variables before you can start working with the Tomcat Web server installed along with the Developer Studio.

Essentially, the protc utility is provided by Progress OpenEdge to perform all the operations that are performed by catalina.sh or catalina.bat scripts provided by Apache Tomcat. The advantage of using the protc utility is that it automatically applies all the configurations you require to run the Tomcat Web server (installed with OpenEdge). For information about executing protc commands, see Executing the protc commands on page 267.

Before you proceed with executing the protc commands to administer the Tomcat Web server, you must create and define the proset.env or proset.bat as follows:

1. Create and open a shell script file, proset.bat (Windows) or proset.env (UNIX), in the OpenEdge work directory ($WRKDIR).

2. In the shell script, define any of the environment variables from Table 52, which you need to customize the Tomcat environment.

   Note: You can see the proset.env or proset.bat examples in Examples of proset script files on page 265.

3. Save and close the shell script file, proset.bat (Windows) or proset.env (UNIX).

Environment variables for the proset shell script file

The following table lists all the environment variables that you can define in the proset shell script file, proset.env (UNIX) or proset.bat (Windows).

Table 71: Environment variables to be defined in proset.bat or proset.env

<table>
<thead>
<tr>
<th>Environment variables</th>
<th>Default</th>
<th>Java property</th>
<th>Variable Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATLINA_HOME</td>
<td>$DLC/servers/tomcat</td>
<td>catalina.home</td>
<td>Specifies the location of the Apache Tomcat instance.</td>
</tr>
<tr>
<td>CATALINA_TMPDIR</td>
<td>$WKDIR/tomcat_wrkdir</td>
<td>java.io.tmpdir</td>
<td>Specifies the location of the temporary directory of Apache Tomcat.</td>
</tr>
<tr>
<td>Environment variables</td>
<td>Default</td>
<td>Java property</td>
<td>Variable Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>---------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>CATALINA_OUT</td>
<td>$CATALINA_HOME</td>
<td>n/a</td>
<td>Specifies the location of Apache Tomcat's stdout and stderr files.</td>
</tr>
<tr>
<td>CATALINA_PID</td>
<td>$CATALINA_HOME/temp</td>
<td>n/a</td>
<td>Specifies the location of Apache Tomcat's pid file, which is used to identify processes running on the Tomcat Web server.</td>
</tr>
<tr>
<td>OPENEDGE_HTTP_PORT</td>
<td>8980</td>
<td>openedge.http.port</td>
<td>Specifies the TCP port for HTTP that OpenEdge uses.</td>
</tr>
<tr>
<td>OPENEDGE_HTTPS_PORT</td>
<td>8943</td>
<td>openedge.https.port</td>
<td>Specifies the TCP port for HTTPS that OpenEdge uses.</td>
</tr>
<tr>
<td>OPENEDGE_SHUTDOWN_PORT</td>
<td>8908</td>
<td>openedge.shutdown.port</td>
<td>Specifies the TCP port for shutting down an Apache Tomcat Web server.</td>
</tr>
<tr>
<td>OPENEDGE_AJP13_PORT</td>
<td>8909</td>
<td>openedge.ajp13.port</td>
<td>Specifies the TCP port used for Tomcat to communicate with Apache/IIS.</td>
</tr>
<tr>
<td>OPENEDGE_WEBAPPSS</td>
<td>$CATALINA_HOME/webapps</td>
<td>openedge.webapps</td>
<td>Specifies a location in the Tomcat Web server where the OpenEdge Web applications are deployed.</td>
</tr>
<tr>
<td>OPENEDGE_WEBLOGS</td>
<td>$WRKDIR</td>
<td>openedge.weblogs</td>
<td>Specifies a location where all the Tomcat related log files are stored.</td>
</tr>
<tr>
<td>Environment variables</td>
<td>Default</td>
<td>Java property</td>
<td>Variable Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------------</td>
<td>--------------------------------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OPENEDGE_USERS_FILE</td>
<td>$CATALINA_HOME/conf/tomcat-users.xml</td>
<td>openedge.users.file</td>
<td>Specifies a location of Apache Tomcat's tomcat-users.xml file, which is used for user management.</td>
</tr>
<tr>
<td>OPENEDGE_REST_JARS</td>
<td>n/a</td>
<td>openedge.rest.jars</td>
<td>Specifies the location where REST Web applications are stored.</td>
</tr>
<tr>
<td>OPENEDGE_ANTIRESOURCE_LOCKING</td>
<td>true</td>
<td>openedge.restANTEsourceLOCKING</td>
<td>When set to true, avoids JAR file locking when resources are accessed inside the JAR files through URLs.</td>
</tr>
</tbody>
</table>

**Examples of proset script files**

The following is an example of the proset.bat file:
### Table 72: proset.bat

```batch
@echo off
rem Customize a personal instance of %DLC%/servers/tomcat
rem This file should be located in your %WRKDIR% directory where
rem it will be found by %DLC%/bin/protc.bat
rem
if not "%WRKDIR%"=="" goto DOSET
echo "environment variable WRKDIR is not defined"
goto END
:DOSET
rem OpenEdge specific Tomcat customizations
rem
set OPENEDGE_HTTP_PORT=8980
set OPENEDGE_SHUTDOWN_PORT=8908
rem set OPENEDGE_HTTPS_PORT=18943
rem set OPENEDGE_AJP13_PORT=18909
set OPENEDGE_WEBAPPS=%WRKDIR%/webapps
rem Set OPENEDGE_REST_JARS only if you want to do compact WAR deployment
rem set OPENEDGE_REST_JARS=%DLC%/rest/lib/*.jar
rem Override standard Tomcat (catalina) variables
rem
set CATALINA_TMPDIR=%WRKDIR%
set CATALINA_OUT=%WRKDIR%/catalina.out
set CATALINA_PID=%WRKDIR%/tomcat.pid
:END
```

The following is an example of the proset.env file:

### Table 73: proset.env

```bash
#!/bin/sh
## Customized a personal instance of $DLC/servers/tomcat
# This file should be located in your $WRKDIR directory where
# it will be found by $DLC/bin/protc
# if [ "${WRKDIR}" = "" ]
then
    echo "environment variable WRKDIR is not defined"
    exit 1
fi
## OpenEdge specific Tomcat customizations
#OPENEDGE_HTTP_PORT=8980 ; export OPENEDGE_HTTP_PORT
OPENEDGE_SHUTDOWN_PORT=8908 ; export OPENEDGE_SHUTDOWN_PORT
#OPENEDGE_AJP13_PORT=8909 ; export OPENEDGE_AJP13_PORT
# OPENEDGE_HTTPS_PORT=8943 ; export OPENEDGE_HTTPS_PORT
OPENEDGE_WEBAPPS="$WRKDIR/webapps" ; export OPENEDGE_WEBAPPS
# OPENEDGE_REST_JARS="$DLC/rest/lib/*.*.jar" ; export OPENEDGE_REST_JARS
## Override standard Tomcat (catalina) variables
#CATALINA_TMPDIR="$WRKDIR" ; export CATALINA_TMPDIR
CATALINA_OUT="$WRKDIR/catalina.out" ; export CATALINA_OUT
CATALINA_PID="$WRKDIR/tomcat.pid" ; export CATALINA_PID
## Add JAVA options to what is set by catalina.sh and setenv.sh
#JAVA_OPTS="" ; export JAVA_OPTS
```
Executing the protc commands

You access the protc utility through the Proenv command-line utility in Start > All Programs > OpenEdge > Proenv. The following example illustrates how to start the Tomcat Web server using the protc utility:

```
protc start
```

The following table lists all the protc commands that you can use to administer the Tomcat Web server.

Table 74: protc commands

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>help</td>
<td>Displays all the protc commands.</td>
</tr>
<tr>
<td>run</td>
<td>Starts the Tomcat Web server in the current window.</td>
</tr>
<tr>
<td>start</td>
<td>Starts the Tomcat Web server in a new window.</td>
</tr>
<tr>
<td>stop</td>
<td>Stops the Tomcat Web server.</td>
</tr>
<tr>
<td>configtest</td>
<td>Runs a basic syntax check on the server.xml file of the Tomcat Web server.</td>
</tr>
<tr>
<td>version</td>
<td>Displays the version of the Tomcat Web server that is currently running.</td>
</tr>
<tr>
<td>status</td>
<td>Displays the status of the Tomcat Web server that is currently running.</td>
</tr>
<tr>
<td>clean</td>
<td>Deletes the temporary directory (CATALINA_TMPDIR) of the Tomcat Web server.</td>
</tr>
</tbody>
</table>

Tuning the Java container 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 REST message), the Java container might fail to send the Web service 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 container that runs the OpenEdge REST Management Agent.

To increase this allocation pool, set the -Xmxsize 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.

Installing a REST Management Agent on a DMZ server

A DMZ server is a server that does not allow you to open untrusted network ports. Since the AdminServer uses ports that might be interpreted as untrusted, the standard REST Management Agent configuration might not run on such a server. The REST Management Agent supports a remote configuration to work around this issue. The remote configuration installs only enough of OpenEdge on the remote server to run the REST Management Agent. This installation includes the REST Management Agent Java Servlet and the files used to support its local configuration.
The remote REST Management Agent uses a `ubroker.properties` file with a limited set of static startup parameters. You must manually edit this file to change the REST Management Agent instance's static configuration properties.

You can continue to administer the remote REST Management Agent's dynamic run-time properties and deploy and manage REST services by mapping it to an AdminServer on the Intranet behind the DMZ server.

To run a remote REST Management Agent configuration:

1. Install the remote REST Management Agent, `oerm.war`, on the DMZ server. Manually edit the static startup parameters in its `ubroker.properties` file and install the REST Management Agent as a Web application in the local Web server's Java container.

2. Start the REST Management Agent in the Web server's Java container, and verify that a browser can access its URL from the Internet.

3. Connect to an AdminServer on the Intranet and configure a "remote" REST Web application instance using OpenEdge Management/OpenEdge Explorer or the RESTMAN utility by specifying the remote REST Management Agent's URL. For details, see the OpenEdge Management/OpenEdge Explorer online help. The Intranet AdminServer creates a REST Management Agent's mapping entry in its local `ubroker.properties` file with the URL of the remote REST Management Agent's administration REST service.

4. Use OpenEdge Management/OpenEdge Explorer or the RESTMAN utility to manage the remote REST Management Agent's dynamic run-time properties and deploy and manage the REST Management Agent's REST services.

   OpenEdge Management or OpenEdge Explorer connects to the Intranet AdminServer where you mapped the REST Management Agent, and then connects to the remote REST Management Agent's REST administration service. When OpenEdge Management or OpenEdge Explorer sends an operation to the Intranet AdminServer, the AdminServer turns it into a REST message and forwards it to the REST Management Agent's REST administration service which executes the operation.

Managing REST Management Agent

The REST Management Agent is a Java servlet running within a Java container. You can manage the REST Management Agent using the Unified Broker framework, including OpenEdge Management, OpenEdge Explorer, and the RESTMAN utility. The REST Management Agent serves as the gateway and management engine for your Web service. It supports all required Web service administration, and manages the exchange of Web service requests and responses between the REST Web application and Web service clients at runtime.

You must configure the REST Management Agent before you can deploy Web services on them. This section contains the following:

- [REST Management Agent administration architecture](#) on page 269
- [Starting and testing a REST Management Agent](#) on page 273
- [Managing a REST Management Agent](#) on page 274

The REST Management Agent includes extensive security features. For more information on security configuration, see [For more information on the RESTMAN utility commands](#), see the RESTMAN utility commands, see [Command and Utility Reference](#) on page 357.
REST Management Agent administration architecture

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

- REST Management Agent administration prerequisites on page 269
- REST administration architecture on page 270
- Web server and Java container configurations on page 271
- REST Management Agent directory structure on page 271

REST Management Agent administration prerequisites

You use OpenEdge Management, OpenEdge Explorer, or the RESTMAN command-line utility to manage the REST Management Agent. However, before you can manage the REST Management Agent, you must complete the following requirements:

- On the system on which the Java container is running, install the REST Management Agent component of OpenEdge as per the instructions in Installing and Configuring REST Management Agent on page 261.
- On the system on which the AdminServer (which administers the REST Management Agent) is running, configure the REST section of the ubroker.properties file to point to the correct REST Management Agent URL. Refer to the following section in the ubroker.properties file:

```
[REST]
  location=Remote
  logFile=C:\OpenEdge\WRK\wsa.log
  proxyHost=
  proxyPassword=
  proxyPort=0
  proxyUsername=
  restUrl=
  webServerAuth=1
  workDir=@{WorkPath}

[REST.restmgr1]
  restUrl=http://localhost:8980/oerm
```

The value of the restUrl must be configured to point to the location of the REST Management Agent. In the above example, the URL is set to indicate that the REST Management Agent (oerm.war) resides in the local computer and is assigned the port 8980.
**REST administration architecture**

The following figure is a general illustration of the components in an implementation of OpenEdge REST Web services.

**Figure 12: OpenEdge REST Web services architecture**

As illustrated in the above figure, the components of an OpenEdge REST Web application deployment can be grouped in four distinct functional areas:

- **AppServers** — The ABL business logic that is accessed by OpenEdge REST Web applications resides on one or more AppServers.

- **Web Server** — OpenEdge REST Web applications are Web Application Archive (WAR) files that reside in a Java Servlet Container on a Web server. OpenEdge REST Web applications interpret incoming HTTP requests from REST client applications and return results from ABL business logic executed by AppServers.
The Web server's Java Servlet Container is also the host of the OpenEdge REST Management Agent, which is an OpenEdge REST Web application that you can optionally install. It enables the management of other of the OpenEdge REST Web applications through OpenEdge utilities.

- **Management Tools** — The interfaces to the REST Management Agent on the Web server are the command line utility, RESTMAN, and the browser-based OpenEdge Explorer/OpenEdge Management tools.
- **Clients** — OpenEdge REST clients can reside on a variety of devices, including servers, desktop, and mobile devices.

### Web server and Java container configurations

The REST Management Agent is implemented as a Java servlet that runs in the context of a Java container as a Web application.

When the REST Management Agent runs within the context of a Java container, several REST Web application instances can run within the context of the REST Management Agent. In addition to the REST Web applications, other applications can run within the Java container, including additional REST Management Agent instances.

You must ensure that all the REST Web applications reside in the same computer as the Web server. And whenever you import or copy a REST Web application (WAR) to the Web Server, ensure that the Web server is configured to unpack and auto-deploy the applications.

For instance, if you are using a Tomcat Web server, you must ensure that the `server.xml` (in the `...\apache-tomcat-7.0.29\conf`) is configured to handle unpackaging and auto-deployment as follows:

```xml
<Host name="localhost" appBase="webapps"
    unpackWARs="true" autoDeploy="true">
```

The `server.xml` file settings denote that the Tomcat Web server and all the REST Web applications reside in the same computer; hence, the `name` is set to `localhost`. Moreover, the values of `unpackWARs` and `autoDeploy` must be set to `true`. With this, whenever you import or copy a REST Web application (WAR) to the Tomcat Web server, the Web server automatically unpackages the WAR file and deploys it on the Web server.

### REST Management Agent directory structure

To configure a REST Management Agent, it is necessary to understand its directory structure. The following figure shows the default directory structure or the REST Management Agent:
Table 75: Directory structure of the REST Management agent as a Java container Web application

<table>
<thead>
<tr>
<th>Directory Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>./REST-management-agent-directory</td>
<td>Includes optional static HTML pages</td>
</tr>
<tr>
<td>META-INF/MANIFEST.MF</td>
<td>Web application descriptor file</td>
</tr>
<tr>
<td>WEB-INF/appSecurity-annonymous.xml</td>
<td>Security configuration model for a REST Web application</td>
</tr>
<tr>
<td>WEB-INF/appSecurity-basic-local.xml</td>
<td>Security configuration model for a REST Web application</td>
</tr>
<tr>
<td>WEB-INF/appSecurity-container.xml</td>
<td>Security configuration model for a REST Web application</td>
</tr>
<tr>
<td>WEB-INF/appSecurity-form-local.xml</td>
<td>Security configuration model for a REST Web application</td>
</tr>
<tr>
<td>WEB-INF/mvc-dispatch-context.xml</td>
<td>Model for dispatching requests to the appropriate handlers</td>
</tr>
<tr>
<td>WEB-INF/user.properties</td>
<td>User authentication and authorization information</td>
</tr>
<tr>
<td>WEB-INF/web.xml</td>
<td>Web application descriptor file</td>
</tr>
<tr>
<td>WEB-INF/lib/restman.jar</td>
<td>List of jar files associated with the Web Application</td>
</tr>
<tr>
<td>WEB-INF/classes/</td>
<td>Java .class files added as required by the developer</td>
</tr>
<tr>
<td>WEB-INF/oerm.data/</td>
<td>List of all the REST Web applications (WAR) deployed and managed by the REST Management Agent (oerm)</td>
</tr>
</tbody>
</table>

The following lists the files that you can use to configure or set your REST Web application (REST Management Agent or any REST Web application):

- **web.xml**: This is a REST Web application descriptor file.
  
  You edit this file when choosing a security configuration and when editing the REST Web application URI. For more information, see Choosing and applying a security configuration on page 312 and the above figure.

- **appSecurity-annonymous.xml, appSecurity-basic-local.xml, appSecurity-container.xml, appSecurity-form-local.xml**: These are the security configuration models for a REST Web application.
  
  You use these files when you want to modify user roles and privileges. For more information, see Security configuration models on page 308.

- **users.properties**: This file consists of all the Web server related user authentication and authorization information. You use this file when adding or modifying a user. For more information, see Adding and modifying a user on page 313 and Setting hashed and salted passwords using Jacksum on page 315.

- **oerm.log**: All the REST Management Agent related transactions are logged in this file.

- **oerm.properties**: This file consists of the definitions for all the REST Management Agent properties.

- **default.prop**: This file consists of the definitions for all the REST Web applications run-time properties.
Starting and testing a REST Management Agent

Once you create REST Management Agent, you can start and test them. This section describes the following:

• Starting the REST Management Agent on page 273
• Testing a REST Management Agent on page 273

Starting the REST Management Agent

Whenever you edit the `ubroker.properties` file for a REST Management Agent and its instances, you must restart the AdminServer. And, whenever you edit the `web.xml` file for a REST Management Agent and its instances, you must restart the REST Management Agent.

To start and run the REST Management Agent:

1. Ensure that your Java container is running.
2. Follow the instructions for your specific Java container and start the REST Management Agent as a Web application. See your Java container vendor documentation on starting and running a Web application.

   Note: If you are using the Tomcat Web server, the REST Management Agent and its associated instances start when you run the Tomcat Web server.

   The only difference between the REST Management Agent and the other Web applications is that the REST Management Agent is automatically enabled when started, whereas the REST Web applications need to be manually enabled before they can be used.

   For more information on configuring the REST Management Agent on a Tomcat Web server, see the Installing a REST Management Agent into a Java container on page 262.

Testing a REST Management Agent

After you create and start a REST Management Agent, you can test it to see if it has started properly.

To test a REST Management Agent:

1. Confirm that the REST Management Agent started correctly by checking for the presence of the files and directories in the following table:

<table>
<thead>
<tr>
<th>Check for the REST Management Agent's . . .</th>
<th>which resides in the directory. . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log file (<code>Agent-name.log</code>)</td>
<td><code>./Agent-name/WEB-INF/Agent-name.data/</code></td>
</tr>
<tr>
<td>Default REST Management Agent properties (<code>Agent-name.properties</code>)</td>
<td>By Default: <code>./oerm/WEB-INF/oerm.data/</code></td>
</tr>
<tr>
<td>Default REST Web application properties file (<code>default.props</code>)</td>
<td></td>
</tr>
<tr>
<td>Holding deployed REST Web applications (WARs)</td>
<td></td>
</tr>
</tbody>
</table>

2. Browse the URL `http://host-name:port-number/oerm/index.html` to see whether the URL is activated.
Managing a REST Management Agent

You manage a REST Management Agent by setting its properties and monitoring its statistics. This section describes the following:

- **Working with properties of a REST Management Agent** on page 274
- **Working with statistics of a REST Management Agent** on page 275
- **Configuring the REST Management Agent for its REST Web application** on page 275

### Working with properties of a REST Management Agent

The REST Management Agent's properties reside in the `oerm.properties` file. By default, the `oerm.properties` file is located in `<web server directory>\oerm\WEB-INF\oerm.data\`. The following are the properties you can review, modify, or reset for a REST Management Agent:

- **adminEnabled** — Specifies whether administrator privileges are in effect, either `enabled` or `disabled`.
- **description** — Specifies the description of the REST Management Agent instance.
- **logAppend** — Specifies whether to start a new log file each time the server is restarted, either `true` or `false`.
- **logEntryTypes** — Specifies a list of logging entry types separated by commas.
- **logFile** — Specifies the path and name of the REST Management Agent log file. This is a read-only property.
- **loggingLevel** — Specifies the amount of information to be written to the broker log.
- **managerPropertiesURI** — Specifies the identifier for the REST Management Agent resource. This is a read-only property.
- **name** — Specifies the name of the REST Management Agent.
- **propertiesFileName** — Specifies the path and name of the `oerm.properties` file.
- **webAppEnabled** — Controls the REST Management Agent's ability to accept and process requests to any of the REST Web applications deployed on it.

To work with the REST Management Agent properties using the RESTMAN utility you use the following:

- **getprops**: To display all the properties of the REST Management Agent. For more information, see **RESTMAN getprops (Management Agent)** on page 384.
- **setprops**: To set one of the properties of the REST Management Agent. For more information, see **RESTMAN setprops (Management Agent)** on page 393.

You can also review and modify the REST Management Agent's properties using the OpenEdge Management or OpenEdge Explorer consoles. For detailed descriptions of managing properties using the OpenEdge Management or OpenEdge Explorer, see the **OpenEdge Management and OpenEdge Explorer: Configuration**.
Working with statistics of a REST Management Agent

You can view and reset run-time statistics for a selected REST Management Agent. Each REST Management Agent instance begins accumulating these statistics, such as **StartTime**. The **StartTime** indicates the last time the Agent instance was started in the Java container.

You can also reset the statistics at any point during REST Management Agent execution, so that the statistics begin accumulating again from a new Start Time.

The following are the statistics you can view or reset for a REST Management Agent:

- **Active Requests** — Requests currently queued and being acted upon by the REST Management Agent
- **Authentication Errors** — Total number of authentication errors
- **HTTP Request Errors** — Requests received from the HTTP listener, including administrative requests, and REST Management Agent requests
- **HTTP Requests** — Total requests received from the HTTP listener, including administrative requests, and REST requests
- **Method not allowed errors** — Total errors returned by the REST Management Agent, with error counts broken out at the bottom of the list for each of several error categories.
- **REST Disabled errors** — calls while REST Management Agent is disabled
- **Services disabled** — REST Web applications deployed on this REST Management Agent that are disabled from client access
- **URL not found errors** — calls made to unidentified resources
- **Creation Time** — Date and time that counting began for the statistics
- **StartTime** — Start date and time of the REST Management Agent instance

To work with the REST Management Agent statistics using the RESTMAN utility you use the following:

- **getstats**: Displays the statistics for a REST Web application, such as Number of Successful User Requests and Average Response Time for User Requests. For more information, see RESTMAN getstats (Management Agent) on page 386.
- **resetstats**: Resets the statistics of a REST Web application. For more information, see RESTMAN resetstats (Management Agent) on page 391.

You can also view and reset the REST Management Agent's properties using the OpenEdge Management or OpenEdge Explorer consoles. For detailed descriptions of managing statistics using the OpenEdge Management or OpenEdge Explorer, see the OpenEdge Management and OpenEdge Explorer: Configuration.

Configuring the REST Management Agent for its REST Web application

Additional configuration settings exist for each REST Management Agent instance that affect the deployment of REST Web applications. These include changing the default values for the properties of REST Web application that you deploy on a REST Management Agent.

When you first run a REST Management Agent, it creates a **default.props** file that contains a collection of properties assigned to each REST Web application deployed on the REST Management Agent. You can configure a REST Management Agent property for its REST Web application by using the RESTMAN utility functions, **setdefault** and **resetdefaults**.

For more information on the RESTMAN utility commands, see Command and Utility Reference on page 357.
Security Configurations for a REST Management Agent

OpenEdge supplements the Java container's security with the industry-recognized Spring security framework. Spring security is a customizable authentication and access-control framework. It is one of the industry standards for securing Spring-based applications. For more information on Spring security framework, see http://docs.spring.io/spring-security/site/docs/4.0.0.RC1/reference/htmlsingle/.

OpenEdge REST Management Agent is a Web application, just as a REST Web application. Therefore, the security configurations for a REST Management Agent and a REST Web application are the same. To apply security configurations on your REST Management Agent, see REST Management Agent and REST Web Application Security Configurations on page 307.
This chapter describes the REST Web applications related administrative tasks.

For details, see the following topics:

- Managing REST Web applications
- Security configurations for a REST Web application
- Managing REST Web applications in the absence of the REST Management Agent
- Managing Web server deployed REST Web applications using the REST Management Agent

Managing REST Web applications

The following sections describe how to manage a REST Web application:

- Initializing a REST Web application's runtime properties on page 278
- Deploying a REST Web application on page 278
- Enabling a REST Web application on page 279
- Generating a REST Web application's WAR and PAAR files on page 279
- Republishing a REST Web application on page 279
- Unpublishing a REST Web service of a REST Web application on page 280
- Monitoring and tuning REST Web application on page 280
- Administering a deployed and enabled REST Web application on page 281
Initializing a REST Web application's runtime properties

Each REST Web application has a set of default runtime properties in its runtime.props file, which by default is in, \...\application-name\WEB-INF\adapters. You can modify the properties using OpenEdge Management, OpenEdge Explorer, or the RESTMAN utility.

**Note:** Except for serviceLoggingLevel and serviceLoggingEntryTypes, you must edit all the other properties only when the application is in a disabled state.

To initialize and work with the REST Web application's properties using the RESTMAN utility, use the following:

- **getprops:** To display all the properties. For more information, see the RESTMAN getprops (Application) on page 383.

- **setprops:** To set a property. For more information, see RESTMAN resetstats (Application) on page 389.

  **Note:** You must not set the serviceAvailable property using the setprops command as that might lead to errors. You must set the property using RESTMAN enable and RESTMAN disable. For more information about the commands, see RESTMAN enable on page 382 and RESTMAN disable on page 381.

- **resetprops:** To reinitialize property values to the current values in the REST Management Agent's default.props file. For more information, see RESTMAN resetprops on page 388.

For detailed descriptions of each REST Web application's properties see Command and Utility Reference on page 357. For more information about using OpenEdge Management or OpenEdge Explorer to initialize properties, see the OpenEdge Management and OpenEdge Explorer: Configuration manual, or the Help for OpenEdge Management or OpenEdge Explorer.

Deploying a REST Web application

You can use OpenEdge Management, OpenEdge Explorer, or the RESTMAN utility's deploy function to deploy or redeploy a REST Web application.

**Note:** When you deploy a REST Web application, the REST Management Agent waits for the application context to start. By default, it waits for up to 1 minute. If the application context does not start as expected, the REST Management Agent does not deploy your application.

If a deployment fails, you can increase the maxDeployWaitTime context parameter, value defined in the REST Management Agent's web.xml.
For more information about using RESTMAN to deploy a REST Web application, see RESTMAN deploy on page 381. For more information about using OpenEdge Management or OpenEdge Explorer to deploy a REST Web application, see the OpenEdge Management and OpenEdge Explorer: Configuration manual, or the OpenEdge Management or the OpenEdge Explorer Help.

Enabling a REST Web application

After you deploy a REST Web application, you must enable it to make it available to incoming client requests. You can enable a REST Web application using OpenEdge Management, OpenEdge Explorer, or RESTMAN utility's enable function.

For more information on using the RESTMAN utility to enable a REST Web application, see the RESTMAN enable on page 382. For more information on using the OpenEdge Management or OpenEdge Explorer to enable a REST Web application, see the OpenEdge Management and OpenEdge Explorer: Configuration manual, or the Help for OpenEdge Management or OpenEdge Explorer.

Generating a REST Web application's WAR and PAAR files

Using Progress Developer Studio for OpenEdge, you can export a REST or Mobile Web application as a WAR file and then deploy the REST or Mobile Web application on your Web server. Similarly, you can also export the REST or Mobile services of a REST or a Mobile Web application as PAAR files and then deploy the REST or Mobile services (PAARs) on your REST or Mobile Web application's instance on the Web server.

For automating Progress Developer Studio for OpenEdge to generate WAR and PAAR files, use the RESTGEN utility. For more information on RESTGEN utility, see RESTGEN on page 377.

You can use the RESTGEN utility to generate a ZIP file with REST or Mobile services, and then use the ZIP file as an input file in the RESTMAN republish utility. For more information on republishing a REST Web application, see Republishing a REST Web application on page 279.

Republishing a REST Web application

In a development environment, you must use the RESTMAN utility's republish command to redeploy (that is, republish) a REST Web application that is deployed on a Web server. The republish command redeploys only the selected files that you specify in the command to their corresponding locations in the REST Web application's folder on the Web server.

Typically, when REST Web application developers have to redeploy and test their applications after updating a few files, such as add or modify a REST Web service (.paar) at %CATALINA_BASE%/webapps/app-name/WEB-INF/adapters/app-name, they use the RESTMAN utility's republish command, not the deploy command.

Using the RESTMAN utility's republish function helps:

- **Reduce network latency**: The republish command only redeploys the changed set of the REST Web application, not all the files of the Web application (which the deploy function does). This reduces network latency.

- **Avoid Locked JAR errors (in Windows)**: Sometimes, the deploy command fails to redeploy a REST Web application because the Web server encounters a locked JAR file because of the way file-locking is implemented in Windows. In such a case, we must use the republish command to successfully redeploy the REST Web application. The republish command deploys only the selected files you specify in the command instead of deploying all the REST Web application files, which helps avoid any locked JAR errors.
**Note:** The republish function throws an error when you try to redeploy a REST Web application that does not have the same folder structure as that of the application instance that exists in the Web server.

If you receive errors while using the republish option, use the RESTMAN utility's deploy function to deploy a fresh instance of your REST Web application. For more information about the deploying a fresh instance of a REST Web application, see Deploying a REST Web application on page 278.

For more information about using RESTMAN to republish a REST Web application, see RESTMAN republish (Application) on page 389. For more information about using OpenEdge Management or OpenEdge Explorer to deploy a REST Web application, see the OpenEdge Management and OpenEdge Explorer: Configuration manual, or the OpenEdge Management or the OpenEdge Explorer Help.

**Unpublishing a REST Web service of a REST Web application**

You can use the unpublish command of the RESTMAN utility to unpublish or delete a Web service (paar file) from a deployed REST Web application in the Web server.

For more information about using RESTMAN to republish a REST Web application, see RESTMAN unpublish (Application) on page 394.

**Monitoring and tuning REST Web application**

To monitor a REST Web application, inspect its statistics. To tune it, modify its properties.

You can monitor a REST Web application using the OpenEdge Management, OpenEdge Explorer or RESTMAN utility.

The following is the list of REST Web application statistics:

- **Adapter errors** — Total number of REST failures
- **Application errors** — Total number of errors related to REST Web applications
- **Average response time** — Average response time of each request response
- **Failed admin logins** — Total number of failed administrator logins
- **Failed admin requests** — Total number of administrator requests that failed due to missing request information
- **Failed app server connections** — Total number of failed AppServer connections in the connection pool
- **Failed user logins** — Total number of failed user logins
- **Failed user requests** — Total number of requests that failed due to missing request information
- **Max response time** — Maximum response time of each request response
- **Successful admin logins** — Total number of successful administrator logins
- **Successful admin requests** — Total number of successful administrator requests
- **Successful app server connections** — Total number of successful AppServer connections in the connection pool for this REST Web application
- **Successful user logins** — Total number of successful user logins
- **Successful user requests** — Total number of successful user requests handled by the selected REST Web application
• **Creation time** — Date and time that counting began for the statistics

• **Start time** — Start date and time of the REST Web application

You use the following RESTMAN utility commands to work with REST Web application statistics:

• **getstats**: Displays the statistics for a REST Web application, such as Number of Successful User Requests and Average Response Time for User Requests. For more information, see RESTMAN getstats (Application) on page 384.

• **resetstats**: Resets the statistics of a REST Web application. For more information, see RESTMAN resetstats (Application) on page 389.

You can also work with REST Web application statistics using OpenEdge Management or OpenEdge Explorer. For more information on the statistics of a REST Web application, see the OpenEdge Management and OpenEdge Explorer: Configuration manual or the OpenEdge Management or OpenEdge Explorer online help.

If the REST Web application uses a NameServer and is session-free, it has a connection pool, with properties you can set. If the Web service is session-managed, it can be set to store multiple requests to the service or repeat them. For more information, see the Reference to OpenEdge Web Service Properties on page 343 and the OpenEdge Management or OpenEdge Explorer online help.

**Administering a deployed and enabled REST Web application**

Once a REST Web application is deployed and enabled, you can administer it as described in the following table.

### Table 76: Actions for administering a deployed and enabled REST Web application

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable and enable</td>
<td>Makes the REST Web application unavailable to Web service clients; makes it available again</td>
</tr>
<tr>
<td>Get, set, reset properties</td>
<td>Provides the value of a REST Web application’s properties; sets or resets one of the REST Web application’s properties</td>
</tr>
<tr>
<td>Get and reset statistics</td>
<td>Displays the value of each REST Web application statistic; reinitializes the statistics</td>
</tr>
<tr>
<td>Query</td>
<td>Provides the REST Web application’s status (enabled or disabled), target URI and statistics URI, and relevant application properties</td>
</tr>
<tr>
<td>Undeploy</td>
<td>Removes the files created when the REST Web application was deployed</td>
</tr>
</tbody>
</table>

To perform these functions, you can use OpenEdge Management, OpenEdge Explorer, or the RESTMAN utility. For more information on REST Web application administration using the RESTMAN utility, see Using the RESTMAN Utility on page 303 and Command and Utility Reference on page 357.

You can also administer REST Web applications using OpenEdge Management or OpenEdge Explorer. For more information, see the OpenEdge Management and OpenEdge Explorer: Configuration manual or the OpenEdge Management or OpenEdge Explorer online help.
Understanding REST Web application URI construction

The REST Web application developer defines the URI path during the REST Web application development. Users access a REST Web application using the URI. For more information on defining a Web application URI, see the Progress Developer Studio for OpenEdge Online help.

This section helps you understand the different components of the Rest Web application URI:

Table 77: Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>http[s]://host:port/rest-app-name/url-pattern</td>
</tr>
</tbody>
</table>

- **host**
  Specifies the location of the host computer on which the Web server resides.
  In the case of REST Web applications, host must be localhost as it is mandatory that the Web server and the REST Web applications reside in the same computer.

- **port**
  Specifies the port number assigned to the Web server.

- **url-pattern**
  Specifies the URI pattern you supplied for the REST Web application.
  For instance, if you are using a Tomcat Web server, you define the value for url-pattern using the `<url-pattern>` tag in the `web.xml` (in the `...\apache-tomcat-7.0.29\conf`).

- **rest-app-name**
  Specifies the name of the REST Web application that is being accessed.

Typically, url-pattern, and rest-app-name are assigned aliases (actual values) during REST Web application development. 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 server and the REST Web application elements and aliases in the following table.
### Table 78: Sample aliases for URL components

<table>
<thead>
<tr>
<th>URL component</th>
<th>Sample alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>localhost</td>
</tr>
<tr>
<td>port</td>
<td>8080</td>
</tr>
<tr>
<td>rest-app-name</td>
<td>customerInfo</td>
</tr>
<tr>
<td>uri-pattern</td>
<td>rest</td>
</tr>
</tbody>
</table>

Given the aliases in the above table, you access the `customerInfo` REST application using the following URI:

```
https://localhost:8080/customerInfo/rest
```

The URI for the REST Web application is defined by the REST Web application developer. Essentially, the application developer constructs and sets up all the input/output parameters (header, body, cookie, query string, path, and form) when developing the ABL REST Web application in Progress Developer Studio for OpenEdge. For more information on constructing the REST Web application URI using REST Expose Editor, see the Progress Developer Studio for OpenEdge online help. As part of URI construction, you must ensure that you are aware of how the URL is encoded when transmitted over the Internet. For more information on HTML URL Encoding, see [http://www.w3schools.com/tags/ref_urlencode.asp](http://www.w3schools.com/tags/ref_urlencode.asp).

### Constructing and debugging REST requests to access a REST Web service

You can identify the information required to invoke an OpenEdge REST Web service by sending a GET request from a REST client in the following URI format:

```
http[s]://<host_name>:<port>/<rest_application_name>/rest
```

The request produces a detailed JSON output that helps you identify how all the REST services are deployed in your REST Web application. This information enables you to debug missing parameters in your REST requests. Additionally, the JSON output helps you identify whether the AppServer is running.

**Note:** As the JSON output consists of all the REST Web application information, viewing the JSON output is restricted to REST Web application users assigned to the `ROLE_PSCDebug` role. However, users assigned to the `ROLE_PSCUSER` role are permitted to retrieve just the AppServer information by sending a GET request in the following URI format:

```
http[s]://<host_name>:<port>/rest/_oeping
```
The JSON output format:

```json
{
    "AppServerStatus":{
        "PingStatus":"true",
        "ABLReturnVal":"",
        "Error": {
            "errorMsg":"[Error message (if any) while running the ABL code]"
            "errorNum":"[Error number associated with the error message]"
        }
    },
    "RESTServices": [
        {"[REST Service1 JSON]"},
        {"[REST Service2 JSON]"},
        ...
        {"[REST ServiceN JSON]"}
    ]
}
```

The `AppServerStatus` section helps you identify the status of the AppServer. The `RESTServices` section lists the status of each REST service deployed in your REST Web application.

**Interpreting the AppServerStatus section**

The `AppServerStatus` section helps you identify the status of the AppServer associated with the REST services in your REST Web application.

The following is a sample JSON output of the `AppServerStatus` section.

```json
{
    "AppServerStatus":{
        "PingStatus":"true",
        "ABLReturnVal":"",
        "Error": {
            "errorMsg":"[Error message (if any) while running the ABL code]"
            "errorNum":"[Error number associated with the error message]"
        }
    },
    "RESTServices": [
        {"..."}
    ]
}
```

The JSON output consists of the following nodes:

- **PingStatus** — If the value of the `PingStatus` node is `true`, it implies that the REST Adapter can ping, or connect with, the AppServer associated with the REST services. If the value is `false`, it implies that there is a problem in connecting with the AppServer, and the `Error` node displays an error message.

You can disable the REST Adapter from performing the ping by setting the value of the `appServerPing` context parameter to `0` in the `web.xml` file of the REST Web application.
• **ABLReturnVal** — The ABLReturnVal node consists of a status message that is returned by an ABL method. This method is configured in the web.xml file of the REST Web application. By default, an empty string is returned with the `ServerStatus()` method of the `OpenEdge.Rest.Admin.AppServerStatus` class. If the REST Adapter is unable to connect with the AppServer, the Error node displays an error message.

You can customize the status message that is returned in the ABLReturnVal node.

To create a customized status message:

1. Create an ABL class and define the `serverStatus()` method to return the desired message. You must save the file in `$WRKDIR`.

   The following is a sample ABL class, MyPing, with the `serverStatus()` method defined to return a character output:

   ```abl
   USING Progress.Lang.*.
   BLOCK-LEVEL ON ERROR UNDO, THROW.
   CLASS MyPing:
      method public char serverStatus( ):
         return "A developer can customize this".
      end.
   END CLASS.
   ```

2. Set the value of the ABLMethod context parameter in the web.xml file.

   ```xml
   <context-param>
      <param-name>ABLMethod</param-name>
      <param-value>MyPing.cls:serverStatus</param-value>
   </context-param>
   ```

   **Note:** The default value of the ABLMethod context parameter is `OpenEdge.Rest.Admin.AppServerStatus.cls:ServerStatus`.

### Interpreting the RESTServices section

The RESTServices section lists the JSON output for each REST service deployed in your REST Web application.

The following is a sample JSON output of two deployed REST services, pingService and BenefitService. It contains information that you require to invoke the BenefitService internal procedures, `DeleteBenefits`, `UpdateBenefits`, `ReadBenefits`, and `CreateBenefits`. 
Note: This sample describes ONLY the `UpdateBenefits` procedure.

```json
{
    "AppServerStatus": {...},
    "RESTServices": {
        "http://localhost:8980/ping/rest/pingService": {...},
        "http://localhost:8980/ping/rest/BenefitService": {
            "Benefits": {
                "Benefits..DeleteBenefits": {},
                "Benefits..UpdateBenefits": {
                    "Method": "PUT",
                    "Resource-Path": "/Benefits",
                    "Http-Request-Type": "application/json",
                    "Http-Response-Type": "application/json",
                    "ABL-Resource-Name": "Benefits.cls",
                    "ABL-Resource-path": "WRKDIR\",
                    "Procedure-Type": "Internal Procedure",
                    "Input": [
                        {
                            "HTTP-Source": "http.body",
                            "ABL-Target": "dsBenefits",
                            "ABL-Type": "DATASET"
                        }
                    ],
                    "Output": [
                        {
                            "HTTP-Target": "http.body",
                            "ABL-Source": "dsBenefits",
                            "ABL-Type": "DATASET"
                        }
                    ]
                }
            }
        }
    }
}
```

Based on the sample JSON output discussed, the following describes the frequently used operations using examples:

- **Constructing the URL to invoke an internal procedure**
  
  Review the JSON output for the `Resource-Path` and `Method` properties. Construct the URL as an HTTP PUT method by appending the value of the `Resource-Path` property to the base URL. The complete URL for accessing the `UpdateBenefits` procedure is `http://localhost:8980/ping/rest/BenefitService/Benefits`.

- **Setting media types for your REST request**
  
  Review the `Http-Request-Type` and `Http-Response-Type` properties. Set your media type to the `application/json` format for your REST request.

- **Sending a value to the ABL input parameter**
  
  Review the `input` section of the JSON output to understand how to send the input parameter values in your REST request to the ABL procedure.

To send values to the `dsBenefits` input parameter of the `DATASET` ABL type, send the JSON representation of the `DATASET` record in your HTTP request body.
• Receiving a value from the ABL output parameter

Review the output section of the JSON to understand how to receive the output parameter values in your REST response from the ABL procedure.

In the example given above, you receive the JSON representation of the `dsBenefits` output parameter in your HTTP response body.

You can also debug the REST Web application’s URL mappings and other definitions during REST Web application development in Progress Developer Studio for OpenEdge. For more information about defining mappings in a REST Web application, see the Progress Developer Studio for OpenEdge online help.

SSL support for REST Web applications

The REST Management Agent 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.

To secure the connection via AppServer protocol between the deployed REST Web application and the AppServer:

• You must obtain and install public key certificates for the host machine in which the REST Management Agent resides.

• The REST Web application 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 `AppServer` or `AppServerDCS`. You set this property, either for a specific service or as the default for services deployed on a given REST Management Agent, by using OpenEdge Management or OpenEdge Explorer or by manually editing the `runtime.props` file of a REST Web application or the `default.props` file of the REST Management Agent. (Note that this property applies to deployed services, not to the REST Management Agent itself; for more information on configuring REST security, see REST Management Agent and REST Web Application Security Configurations on page 307.)

• The AppServer must be SSL-enabled, that is, it must accept SSL requests from the REST Management Agent (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 Management, 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 SSL-enabled AppServer operation on page 45 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 - Security and Auditing.

Enabling the REST Web application for HTTPS client connections

OpenEdge supports SSL-enabled communications between the Web server on which the REST Management Agent runs and a Web server client. To achieve this, you must enable the REST Web application for HTTPS (SSL) connections.

To enable the REST Web application for SSL:

1. Open the REST Web application's `web.xml` file (typically, in the folder `application-name\WEB-INF`).
2. Identify and uncomment the code inside the tag `<!--Certificate jar location-->`.  

OpenEdge Application Server: Administration 287
After uncommenting, your code must look as shown below:

```xml
<!-- Certificate jar location -->
<context-param>
    <param-name>certlocation</param-name>
    <param-value>psccerts-directory-location</param-value>
</context-param>
```

**Note:** The `psccerts` is the certificate shipped with the OpenEdge installation. If you wish to use your own certificate, you must ensure that the path in the `<param-value>` tag points to the directory location of your certificate.

3. Enable and configure the Web server for SSL support.
   This includes installing the Web server digital certificate on the Web server. For more information, see the Web server documentation.

4. Using a text editor, edit the `ubroker.properties` file to set the REST URL (value of the `restUrl` property) to use the HTTPS protocol (instead of HTTP).

5. Save the `ubroker.properties` file.

### SSL-related REST Web application properties

You can set the following properties as defaults for the REST Web application deployed on a given REST Management Agent:

**Note:** The REST Web application properties reside in its `runtime.props` file (typically, in the folder `application-name\WEB-INF`).

- **appServiceProtocol** — Assigns a value of `AppServerS` or `AppServerDCS` to support SSL communication with the AppServer
- **noHostVerify** — Controls whether the REST Web application 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 REST Web application properties, see Reference to OpenEdge Web Service Properties on page 343.

### SSO support for REST Web applications

Single sign-on (SSO) is a property of access control of multiple related systems. SSO validates a user identity that might have been authenticated at another location, but within the same security domain.
One of the primary Web application best practice is for the Web server (that is, Java container) to handle authentication and authorization of the Web application resources. For this purpose, the Spring Security framework was added to the REST Web application build. For more information, see REST Management Agent and REST Web Application Security Configurations on page 307. However, while this is the right way for Web application security to operate, it does not satisfy the ABL AppServer service’s requirements to know who is the current logged-in user and what is the user's login session-id. The lack of user login session coordination between the Web application and the AppServer promotes ABL developers to bypass web application security and attempt handling the application security in the AppServer end as if it were just another on-premise ABL client.

REST Web applications address this issue by enabling SSO. In this case, the REST Web application represents the AppServer client which has successfully authenticated to an external source (that is, the Web server) and is identified by the AppServer using the same identity. In this way you may employ a strong Web server security and operate with the same user identity and login session.

**Enabling the REST Web application for SSO**

To enable the REST Web application for SSO:

1. Identify and open the security configuration you applied to your REST Web application.

   Your REST Web application’s security configuration is based on your performing the instructions in Choosing and applying a security configuration on page 312.

2. In the security configuration file, `appSecurity-XXX.xml`, uncomment the `OEClientPrincipalFilter` bean code snippet to enable SSO support.

   After uncommenting, your `OEClientPrincipalFilter` bean code snippet must look as shown below:

```xml
<!-- The security filter that turns a Spring token into an OpenEdge ClientPrincipal object -->
<b:bean id="OEClientPrincipalFilter"
    class="com.progress.rest.security.OEClientPrincipalFilter">
    <b:property name="enablecp" value="true" />
    <b:property name="domain" value="$Application" />
    <b:property name="roles" value="" />
    <b:property name="authz" value="" />
    <b:property name="expires" value="600" />
    <b:property name="acctinfo" value="" />
    <b:property name="" >
        <b:map>
            <b:entry key="prop-1" value="string1"/>
            <b:entry key="prop-2" value="string2"/>
        </b:map>
    </b:property>
    <b:property>
        <b:property name="ccid" value="true" />
        <b:property name="anonymous" value="true" />
    </b:property>
</b:bean>
```
Optionally, for each of the OEClientPrincipal properties, such as enablecp and domain, you can update the value attribute. To know about the OEClientPrincipal properties, see OEClientPrincipal properties on page 290. The following is a sample with updated properties values:

```xml
<!-- The security filter that turns a Spring token into an OpenEdge ClientPrincipal object -->
<b:bean id="OEClientPrincipalFilter"
      class="com.progress.rest.security.OEClientPrincipalFilter">
    <b:property name="enablecp" value="false" />
    <b:property name="domain" value="sample" />
    <b:property name="roles" value="sample" />
    <b:property name="authz" value="false" />
    <b:property name="expires" value="600" />
    <b:property name="accntinfo" value="true" />
    <b:property name="properties">
        <b:map>
            <b:entry key="prop-1" value="string1"/>
            <b:entry key="prop-2" value="string2"/>
        </b:map>
    </b:property>
    <b:property name="ccid" value="true" />
    <b:property name="anonymous" value="true" />
</b:bean>
```


**Note:** You must restart the Web server for the above security configuration updates to take effect.

### OEClientPrincipal properties

The OEClientPrincipalFilter is a standard Spring security filter bean. It is declared in the Spring security configuration files in which its properties can be customized to override the defaults.

The following table describes all the property names:

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
<th>Datatypes</th>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>enablecp</td>
<td>Specifies if SSO is enabled or disabled.</td>
<td>Boolean</td>
<td>true</td>
<td>true or false</td>
</tr>
<tr>
<td>domain</td>
<td>Specifies Web application's domain name</td>
<td>String</td>
<td>&lt;webapp-name&gt;</td>
<td>ASCII</td>
</tr>
<tr>
<td>roles</td>
<td>Specifies comma-separated list of ROLE names. You use this to perform Role-Based-Authentication (RBA) inside your business application.</td>
<td>String</td>
<td>-</td>
<td>ASCII</td>
</tr>
</tbody>
</table>

---

6. If you are using an anonymous security configuration model (appServer-anonymous.xml) for your REST Web application, you must ensure that the roles property value is left blank because if you do not mention any value for the roles property, the REST Web application assumes the default value, ROLE_ANONYMOUS.
<table>
<thead>
<tr>
<th><strong>Property name</strong></th>
<th><strong>Description</strong></th>
<th><strong>Datatypes</strong></th>
<th><strong>Default</strong></th>
<th><strong>Range</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>authz</td>
<td>Specifies comma-separated list of ROLE names</td>
<td>String</td>
<td>-</td>
<td>ASCII</td>
</tr>
<tr>
<td>expires</td>
<td>Specifies the data expiration time in seconds</td>
<td>String integer</td>
<td>0</td>
<td>0 to n</td>
</tr>
<tr>
<td>acctinfo</td>
<td>Specifies the Spring token account state properties</td>
<td>Boolean</td>
<td>-</td>
<td>true or false</td>
</tr>
<tr>
<td>properties</td>
<td>Specifies a map, a collection of property name entries</td>
<td>Map</td>
<td>-</td>
<td>String property name and value</td>
</tr>
<tr>
<td>ccid</td>
<td>Specifies if CCID is enabled or disabled</td>
<td>Boolean</td>
<td>true</td>
<td>true or false</td>
</tr>
<tr>
<td>&quot;key&quot;</td>
<td>The key to be used for sealing the C-P</td>
<td>String</td>
<td>&quot;&quot;</td>
<td>Cleartext or oech1::xxxx where xxxx is the output of genpassword utility</td>
</tr>
<tr>
<td>&quot;registryFile&quot;</td>
<td>The domain registry file used to seal the C-P NOTE: You need to configure either &quot;key&quot; or &quot;registryFile&quot;</td>
<td>File path</td>
<td>null</td>
<td>generated using the gendomreg utility</td>
</tr>
<tr>
<td>anonymous</td>
<td>Specifies if the AppServer is being called from an anonymous Web application user or not</td>
<td>Boolean</td>
<td>true</td>
<td>true or false</td>
</tr>
<tr>
<td>sealAnonymous</td>
<td>Property to seal the anonymous C-P token using &quot;key&quot; or &quot;registryFile&quot;</td>
<td>Boolean</td>
<td>false</td>
<td>true or false</td>
</tr>
</tbody>
</table>

---

7 The properties property is provided for you to pass any Spring token's state or any static information to the AppServer. For instance, you can include Spring token's state, isAccountLocked and isAuthenticated, properties as entries in your properties property.

Note that, if you are using an anonymous security configuration model (appServer-anonymous.xml), Spring tokens do not get authenticated for any user account; which means that no Spring security state related entries in your properties property is passed to the AppServer.

8 In the case of certain clients where COOKIES are not allowed and REST JSON data services do not perform URL rewriting of session-ids, the clients use CCID functionality (available to other AppServer clients) to obtain the user's login session-id. The CCID support inserts a X-CLIENT-CONTEXT-ID http header in each response message. The client may obtain the value and insert it into the next request's URL as a "JSESSIONID" query option.

9 To use gendomregistry, create a .csv file with domain and domain access code. For example: Create a domain.csv file as follows: acme.com,letmein acme.eu,secret. Use the gendomreg utility to create a registry file. To create a registry file with name as domain.bin, run the command gendomreg domainPasses.csv domains.bin
Pre-authentication filter in REST Web applications

The appSecurity-xxx.xml security configuration file of the REST Web application consists of a PRE-AUTH filter for bypassing the REST authentication process for REST requests that have been pre-authenticated by other applications. The bypass operation is performed if the REST request contains a pre-authenticated ClientPrincipal token in the REST request header. For pre-authenticated requests, the REST Web application performs just the domain authorization test to grant access.

The following OEPreauthfilter bean code snippet is the PRE-AUTH filter format:

```xml
<!-- The Pre-auth security filter for SSO -->
<b:bean id="OEPreauthfilter"
  class="com.progress.rest.security.OERequestHeaderAuthenticationFilter">
  <b:property name="enabled" value="false"/>
  <b:property name="key" value=""/>
  <b:property name="registryFile" value=""/>
  <b:property name="encoding" value="OECP"/>
  <b:property name="headername" value="X-OE-CLIENT-CONTEXT-ID"/>
  <b:property name="authenticationManager" ref="authenticationManager"/>
</b:bean>

<authentication-manager alias="authenticationManager">
  <authentication-provider ref="preauthAuthProvider"/>
</authentication-manager>
<b:bean id="preauthAuthProvider"
  class="com.progress.rest.security.OEPreAuthenticatedAuthenticationProvider">
  <!--
  <b:property name="rolePrefix" value="ROLE_"/>
  <b:property name="enabledAttrName" value="ATTR_ENABLED"/>
  <b:property name="lockedAttrName" value="ATTR_LOCKED"/>
  <b:property name="expiredAttrName" value="ATTR_EXPIRED"/>
  -->
  <b:property name="rolePrefix" value="/"/>
  <b:property name="enabledAttrName" value="AUTH_ENABLED"/>
  <b:property name="lockedAttrName" value="AUTH_LOCKED"/>
  <b:property name="expiredAttrName" value="AUTH_EXPIRED"/>
</b:bean>

Note: Other Progress applications, such as Rollbase®, use ATTR_ENABLED, ATTR_LOCKED, and ATTR_EXPIRED as user state names. It is recommended that you do not change these names.

Enabling the pre-authentication filter

To enable the PRE-AUTH pre-authentication filter:

1. Set the value attribute of the enabled property as true.
2. Validate the ClientPrincipal token against domains:
   - For validating the token against a single domain, set the domain access code as the value attribute of the key property. The domain access code can be clear text or in the oecl::<hex-string> format.

   Note: You can generate the <hex-string> using the genpassword utility. For more information about the genpassword utility, see OpenEdge Getting Started: Installation and Configuration.

   - For validating the token against multiple domains:
1. Create a domain registry file using the Generate Domain Registry `gendomreg.bat` utility that is located in the `$DLC\bin` folder.

2. Uncomment the `registryFile` property in the `OEPreauthfilter` bean code.

3. Set the value attribute of the `registryFile` property as the output file name generated by the `gendomreg.bat` utility.

**Note:** If you set values for both `key` and `registryFile` properties, the `ClientPrincipal` token is validated against multiple domains that are specified by the `registryFile` property value. To validate the token against a single domain, comment out the `registryFile` property in the `OEPreauthfilter` bean code.

---

**Debugging HTTP messages of a REST Web application**

In a production environment, the REST Management Agent enables you to collect (or dump) HTTP request and response messages of your REST Web application in a log file, which you can use for debugging. This section discusses:

- **Enabling HTTP message dumping** on page 293
- **Disabling HTTP message dumping** on page 296

**Enabling HTTP message dumping**

For dumping HTTP request and response messages, you add the `RequestDumperFilter` configurations to your Tomcat Web server instance where your REST Web application is hosted. Then, you enable your REST Web application to allow for its HTTP request and response messages dumping.

**Note:** Enabling HTTP message dumping is a REST Web application performance and security concern. Progress recommends that you enable this only for debugging purposes.

To enable HTTP message dumping:

1. Take a back-up of the Web server's `logging.properties` file and the REST application's `web.xml` file by executing the following commands:

   ```
   cp %DLC%\servers\tomcat\conf\logging.properties %DLC%\servers\tomcat\conf\logging.properties.orig
   cp %CATALINA_BASE%\webapps\<REST_Web_Application_Name>\WEB-INF\web.xml %DLC%\servers\tomcat\conf\web.xml.orig
   ```

   These backup files are used when you want to disable HTTP message debugging.

2. If you are using the Tomcat Web server installed with Progress Developer Studio for OpenEdge, then open the `logging.properties` file from `%DLC%\servers\tomcat\conf\` in a text editor, and locate the following lines of code:

   ```
   ############################################################
   # RequestDumperFilter Configuration.
   # Dumps the HTTP request and response
   ############################################################
   ```
# To see the HTTP request and response dump, uncomment the below lines
# NOTE: To enable dumping of HTTP request & response, you will also need to
# uncomment the OERequestDumperFilter section in the web.xml of the web
# application
# that you want to debug.

#5request-dumper.org.apache.juli.FileHandler.level = FINEST
#5request-dumper.org.apache.juli.FileHandler.directory =
#${openedge.weblogs}
#5request-dumper.org.apache.juli.FileHandler.formatter =
#org.apache.juli.VerbatimFormatter
## The level must be set at least to INFO. If you want to dump the body
also, set it to FINEST
#org.apache.catalina.filters.RequestDumperFilter.level = INFO
#org.apache.catalina.filters.RequestDumperFilter.handlers =
#5request-dumper.org.apache.juli.FileHandler

Uncomment the request-dumper related lines of code. After uncommenting, the code block must be:

```
# RequestDumperFilter Configuration.
# Dumps the HTTP request and response
# NOTE: To enable dumping of HTTP request & response, you will also need to
# uncomment the OERequestDumperFilter section in the web.xml of the web application
# that you want to debug.

#5request-dumper.org.apache.juli.FileHandler.level = FINEST
#5request-dumper.org.apache.juli.FileHandler.directory =
#${openedge.weblogs}
#5request-dumper.org.apache.juli.FileHandler.prefix =
#5request-dumper.org.apache.juli.FileHandler.formatter =
#org.apache.juli.VerbatimFormatter
## The level must be set at least to INFO. If you want to dump the body also, set it
to FINEST
#org.apache.catalina.filters.RequestDumperFilter.level = INFO
#org.apache.catalina.filters.RequestDumperFilter.handlers =
#5request-dumper.org.apache.juli.FileHandler
```

3. If you are using a Tomcat Web server other than the one installed with Progress Developer Studio for
OpenEdge, then you must modify the logging.properties file from $CATALINA_BASE\tomcat\conf\ in a text editor to enable HTTP message dumping.

The following is an example of an edited logging.properties file that you can use to edit your Web
server's logging.properties file:

```
# To this configuration below, lrequest-dumper.org.apache.juli.FileHandler
# also needs to be added to the handlers property near the top of the file
lrequest-dumper.org.apache.juli.FileHandler.level = INFO
lrequest-dumper.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
lrequest-dumper.org.apache.juli.FileHandler.formatter =
org.apache.juli.VerbatimFormatter
# Change the level from INFO to FINEST if you want to dump the HTTP body as well
org.apache.catalina.filters.RequestDumperFilter.level = INFO
org.apache.catalina.filters.RequestDumperFilter.handlers = l
lrequest-dumper.org.apache.juli.FileHandler
```
For more information on configuring the logging.properties file, see the Tomcat documentation at http://tomcat.apache.org/tomcat-7.0-doc/logging.html.

4. Optionally, if you want to dump both the request and response body of the HTTP message, then change the RequestDumperFilter.level from INFO to FINEST in the logging.properties file:

   org.apache.catalina.filters.RequestDumperFilter.level = FINEST

5. Open the web.xml file of your REST Web application from %CATALINA_BASE%\webapps\<REST_Web_Application_Name>\WEB-INF, and locate the following lines of code:

   <!-- USER EDIT: uncomment OERequestDumperFilter section to enable debugging of HTTP Request/Response -->
   <!-- NOTE You must also uncomment RequestDumperFilter Configuration section in $CATALINA_HOME/conf/logging.properties file to redirect the debug messages to ${openedge.weblogs}/request-dumper.<date>.log file. ${openedge.weblogs} is set to OpenEdge WRKDIR by default -->

   <!-- 
   <filter>
   <display-name>OERequestDumperFilter</display-name>
   <filter-name>OERequestDumperFilter</filter-name>
   <filter-class>com.progress.rest.adapters.oe.OERequestDumperFilter
   </filter-class>
   </filter>
   
   <filter-mapping>
   <filter-name>OERequestDumperFilter</filter-name>
   <url-pattern>/*</url-pattern>
   </filter-mapping>
   -->

Uncomment the OERequestDumperFilter related lines of code. After uncommenting, the code-block must be:

   <!-- USER EDIT: uncomment OERequestDumperFilter section to enable debugging of HTTP Request/Response -->
   <!-- NOTE You must also uncomment RequestDumperFilter Configuration section in $CATALINA_HOME/conf/logging.properties file to redirect the debug messages to ${openedge.weblogs}/request-dumper.<date>.log file. ${openedge.weblogs} is set to OpenEdge WRKDIR by default -->

   <filter>
   <display-name>OERequestDumperFilter</display-name>
   <filter-name>OERequestDumperFilter</filter-name>
   <filter-class>com.progress.rest.adapters.oe.OERequestDumperFilter
   </filter-class>
   </filter>
   
   <filter-mapping>
   <filter-name>OERequestDumperFilter</filter-name>
   <url-pattern>/*</url-pattern>
   </filter-mapping>
After enabling the HTTP message dumping, all the HTTP request and response messages will be logged in the `request-dumper.<date>.log` file, which is located in the OpenEdge `WRKDIR` by default. You can use the logs in this file for debugging purposes.

**Disabling HTTP message dumping**

Enabling HTTP message dumping is a REST Web application performance and security concern. Therefore, Progress recommends that you enable this only for debugging purposes, and disable it after debugging.

To disable HTTP message dumping:

1. Stop the Tomcat Web server on which the REST Web application is hosted.

   **Note:** If you are using the Tomcat Web server shipped with Progress Developer Studio for OpenEdge, you can start and stop your Tomcat Web server using the `protc` utility.

2. Replace Web server’s `logging.properties` file and the REST application’s `web.xml` file with its instance that exists from before you updated the files for HTTP message debugging. Executing the following commands:

   ```
   cp %DLC%\servers\tomcat\conf\logging.properties.orig
   %DLC%\servers\tomcat\conf\logging.properties
   cp %CATALINA_BASE%\webapps\<REST_Web_Application_Name>\WEB-INF\web.xml.orig
   %DLC%\servers\tomcat\conf\web.xml
   ```

   Essentially, as part of this step, you are removing all the modifications in the `logging.properties` and the `web.xml` files that you did as part of enabling HTTP message dumping.

**Security configurations for a REST Web application**

OpenEdge supplements the Java container’s security with the industry-recognized Spring security framework. Spring security is a customizable authentication and access-control framework. It is one of the industry standards for securing Spring-based applications. For more information on Spring security framework, see [http://docs.spring.io/spring-security/site/docs/4.0.0.RC1/reference/htmlsingle/](http://docs.spring.io/spring-security/site/docs/4.0.0.RC1/reference/htmlsingle/)

To apply security configurations on your REST Web applications, see **REST Management Agent and REST Web Application Security Configurations** on page 307.

**Managing REST Web applications in the absence of the REST Management Agent**

Progress Software recommends you to administer or manage REST Web applications using the REST Management Agent (`oerm.war`) shipped with the Progress Developer Studio for OpenEdge installation. But, in a scenario in which you must manage REST Web applications in the absence of the REST Management Agent, you can perform the following processes as required by the REST Web application without using the REST Web application:

- **Deploying the REST Web application** on page 297
- **Editing a REST Web application runtime properties** on page 297
• Editing a REST Web application security configurations on page 297
• Enabling a REST Web application using Tomcat's manager on page 298
• Monitoring and maintaining REST Web application log files on page 298

**Note:** In the absence of the REST Management Agent (oerm.war), you cannot manage REST Web applications using OpenEdge Management or OpenEdge Explorer.

## Deploying the REST Web application

The following are two ways of deploying your application in the absence of the REST Management Agent:

**Note:** It is assumed that you have developed a REST Web application and that your Web server automatically unpackages the application resources (refer to Web server and Java container configurations on page 95).

- By using the **Web server native deployer**: Use the Web server native deployer to deploy your REST Web applications. For instance, if you are using the Tomcat Web server, you can use the Tomcat Deployer to deploy your REST Web application. For more information, see the Web server documentation.

- By manually copying the **REST Web application in the Web server**: Copy the REST Web application (WAR) to the Web server deployment directory. For instance, if you are using a Tomcat Web server, copy the REST Web application to the `$CATALINA_HOME/webapps` directory.

  **Note:** If the Web server is already running, you must re-enable the REST Management Agent using Web server’s management console or restart the Web server for the new configuration settings to take effect.

## Editing a REST Web application runtime properties

All the deployed REST Web applications (WAR files) contain a `runtime.props` file, in which all the application run-time properties are defined. To edit the REST Web application run-time properties, you must open and edit the properties in the `runtime.props` (in the location `$CATALINA_HOME/webapps\application-name\WEB-INF\adapters\`). You can also edit the connection information contained the `runtime.props` file.

**Note:** If the Web server is already running, you must re-enable the REST Management Agent using Web server’s management console or restart the Web server for the new configuration settings to take effect.

For information on applications properties, see Reference to OpenEdge Web Service Properties on page 343.

## Editing a REST Web application security configurations

Security configurations of a REST Web application do not get affected by the absence of the REST Management Agent. To apply security configurations on your REST Web applications, see the REST Management Agent and REST Web Application Security Configurations on page 307.
Enabling a REST Web application using Tomcat's manager

After you deploy a REST Web application, you must enable it to make it available to in-coming client requests. You enable the application by setting the REST Web application run-time property, `serviceAvailable`, to 1.

An enabled REST Web application will have the following setting in the `runtime.props` file:

```xml
<bpm:serviceAvailable>1</bpm:serviceAvailable>
```

The `runtime.props` file is in `~/CATALINA_HOME/webapps/application-name/WEB-INF/adapters/`. To disable the application you must set the `serviceAvailable` property to 0.

**Note:** If the Web server is already running, you must re-enable the REST Management Agent using Web server's management console or restart the Web server for the new configuration settings to take effect.

Monitoring and maintaining REST Web application log files

To monitor and maintain the application logs you make use of the following files of a REST Web application:

- **log4j.properties**: You use this file to control the log settings of third-party components. To control the log settings, you must refer to the instructions inside the `log4j.properties` file and uncomment the lines of code as per your requirement. The `log4j.properties` file is in `~/CATALINA_HOME/webapps/application-name/WEB-INF/classes/`

- **runtime.props**: You use this file to control the log setting of the OpenEdge components. The `runtime.props` file is in `~/CATALINA_HOME/webapps/application-name/WEB-INF/adapters/`

The two properties, in the `runtime.props` file, you edit are:

- `serviceLoggingLevel`
- `serviceLogEntryTypes`

If you want to delete or move the log files, you must stop the REST Web application context using the Web server's management console.

For information about these REST Web application properties, see Reference to OpenEdge Web Service Properties on page 343.

Managing Web server deployed REST Web applications using the REST Management Agent

Typically, REST Management Agent (`oerm.war`) manages applications deployed using the REST clients: RESTMAN utility, OpenEdge Management, or OpenEdge Explorer. Whenever the REST clients successfully deploy the REST Web application, all the REST Web application details are entered in the `oermDeployedAppList.xml` file of the REST Management Agent.

When the REST Management Agent is deployed for the first time (refer to Installing and Configuring REST Management Agent on page 261), the `oermDeployedAppList.xml` file is created in `~/CATALINA_HOME/webapp/oerm/WEB-INF/oerm.data/`, and all the REST Web applications that are deployed using the REST Management Agent are listed in it.
For example, if you have deployed an application, RESTApp, using the REST Management Agent (oerm.war), the oermDeployedAppList.xml file gets populated with the RESTApp application details as follows:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Application Name="RESTApp">
  <Description>This is sample description about RESTApp</Description>
  <Status>DISABLED</Status>
  <URI>http://localhost:8080/oerm/applications/RESTApp</URI>
  <PropertiesURI>
    http://localhost:8080/oerm/applications/RESTApp/properties
  </PropertiesURI>
  <StatisticsURI>
  </StatisticsURI>
  <PropFileLoc>
    F:\Softwares\Apache-Tomcat\apache-tomcat-7.0.21\webapps\oerm..\RESTApp\WEB-INF/adapters\runtime.props
  </PropFileLoc>
  <adapterDirLoc>
    F:\Softwares\Apache-Tomcat\apache-tomcat-7.0.21\webapps\oerm..\RESTApp\WEB-INF/adapters
  </adapterDirLoc>
</Application>
</Applications>
```

The following table lists the deployed application details:

**Table 80: Deployed REST Web application details**

<table>
<thead>
<tr>
<th>Application detail</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Name</td>
<td>Name of the application.</td>
<td>String</td>
</tr>
<tr>
<td>Description</td>
<td>Small description about the application.</td>
<td>String</td>
</tr>
<tr>
<td>Status</td>
<td>Current state of the application.</td>
<td>Disabled or Enabled</td>
</tr>
<tr>
<td>URI</td>
<td>URI at which information about the application can be accessed.</td>
<td>Hyperlink</td>
</tr>
<tr>
<td>PropertiesURI</td>
<td>URI at which information about the application properties can be accessed.</td>
<td>Hyperlink</td>
</tr>
<tr>
<td>StatisticsURI</td>
<td>URI at which information about the application statistics can be accessed.</td>
<td>Hyperlink</td>
</tr>
<tr>
<td>PropFileLoc</td>
<td>Fully qualified path of the REST Web application runtime properties file, runtime.props.</td>
<td>Path</td>
</tr>
<tr>
<td>adapterDirLoc</td>
<td>Fully qualified path of the REST Web application adapters directory.</td>
<td>Path</td>
</tr>
</tbody>
</table>

---

10 This is the only application detail that is dynamically updated whenever a REST client enables or disables the application.
If you want to manage a Web server deployed REST Web application using the REST Management Agent, you must enter the application details in the oermDeployedAppList.xml file of the REST Management Agent (oerm.war) for it to recognize the application as one of the REST Management Agent managed application.

For example, if you want to manage a Web server deployed REST Web application, RESTApp1, using the REST Management Agent, open the oermDeployedAppList.xml file, copy and paste the sample code snippet, and then fill the, RESTApp1, application details as shown below:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Application Name="RESTApp">
    <Description>This is sample description about RESTApp</Description>
    <Status>DISABLED</Status>
    <URI>http://localhost:8080/oerm/applications/RESTApp</URI>
    <PropertiesURI>
        http://localhost:8080/oerm/applications/RESTApp/properties
    </PropertiesURI>
    <StatisticsURI>
    </StatisticsURI>
    <PropFileLoc>
        F:\Softwares\Apache-Tomcat\apache-tomcat-7.0.21\webapps\oerm..\RESTApp\WEB-INF/adapters\runtime.props
    </PropFileLoc>
    <adapterDirLoc>
        F:\Softwares\Apache-Tomcat\apache-tomcat-7.0.21\webapps\oerm..\RESTApp\WEB-INF/adapters
    </adapterDirLoc>
</Application>

<Application Name="RESTApp1">
    <Description>This is sample description about RESTApp1</Description>
    <Status>ENABLED</Status>
    <URI>http://localhost:8080/oerm/applications/RESTApp1</URI>
    <PropertiesURI>
        http://localhost:8080/oerm/applications/RESTApp1/properties
    </PropertiesURI>
    <StatisticsURI>
    </StatisticsURI>
    <PropFileLoc>
        F:\Softwares\Apache-Tomcat\apache-tomcat-7.0.21\webapps\oerm..\RESTApp1\WEB-INF/adapters\runtime.props
    </PropFileLoc>
    <adapterDirLoc>
        F:\Softwares\Apache-Tomcat\apache-tomcat-7.0.21\webapps\oerm..\RESTApp1\WEB-INF/adapters
    </adapterDirLoc>
</Application>
</Applications>
```

Note: The sample RESTApp application is in the DISABLED state and the RESTApp1 application is in the ENABLED state. For an enabled REST Web application, the serviceAvailable run-time property is set to 1. For information about these REST Web application properties, see the Reference to OpenEdge Web Service Properties on page 343.
For all the REST Web applications listed in the oermDeployedAppList.xml file, you can use any of the supported REST clients (RESTMAN, OpenEdge Explorer, or OpenEdge Management) to manage your application.
Using the RESTMAN Utility

The RESTMAN utility is a command-line utility for administering the REST Management Agent and the REST Web applications. The REST Management Agent deploys the REST Web applications on the Web server. The functions provided by the RESTMAN utility duplicate their OpenEdge Management and OpenEdge Explorer equivalents. Essentially, RESTMAN utility is a remote Web application and the three clients that can access the application features are OpenEdge Management, OpenEdge Explorer, or RESTMAN utility.

The RESTMAN utility operates in a similar fashion to the WSAMAN utility in that it communicates using Remote Method Invocation (RMI) with the OpenEdge AdminServer, which communicates with the OpenEdge Management Agent using the REST API. You use the RESTMAN utility from the Proenv command-line utility shipped as part of your OpenEdge installation.

For details, see the following topics:

- Syntax
- Functions for managing a REST Management Agent
- Functions for managing a REST Web application

Syntax

Each function of the RESTMAN utility has syntax similar to the following:

Table 81: Syntax

```
restman [ -option]... -function
```
[ -option] . . .

One or more options, where each option consists of a name (for example, -appname) and a value (for example, -value).

function

The name of the function (for example, getdefaults).

For more information on the functions of the RESTMAN utility, see Command and Utility Reference on page 357.

Functions for managing a REST Management Agent

Certain functions of the RESTMAN utility are specific to the REST Management Agent and enable you to manage it. These are summarized in the following table.

Table 82: REST Management Agent functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESTMAN getdefaults on page 383</td>
<td>Displays the default properties associated with the REST Management Agent.</td>
</tr>
<tr>
<td></td>
<td>All the properties of the REST Management Agent are in its default.props file that is used to initialize the runtime.props file of each REST Web application deployed by the REST Management Agent on the Web server.</td>
</tr>
<tr>
<td>RESTMAN getprops (Management Agent) on page 384</td>
<td>Displays the current value of the properties of REST Management Agent that can be changed while it is running.</td>
</tr>
<tr>
<td>RESTMAN getstats (Management Agent) on page 386</td>
<td>Displays statistics for the REST Management Agent.</td>
</tr>
<tr>
<td>RESTMAN list on page 386</td>
<td>Displays the list of REST Web applications that have been deployed by the REST Management Agent on the Web server.</td>
</tr>
<tr>
<td>RESTMAN query (Management Agent) on page 387</td>
<td>Displays the following characteristics and properties of the REST Management Agent:</td>
</tr>
<tr>
<td></td>
<td>• Status of the Agent (running, not running, or status not known).</td>
</tr>
<tr>
<td></td>
<td>• REST Administration and the REST Services status. Displays whether they are enabled or disabled.</td>
</tr>
<tr>
<td></td>
<td>• URI information: Displays the properties URI, statistics URI, and the Applications URI associated with the REST Management Agent.</td>
</tr>
<tr>
<td>RESTMAN resetdefaults (Management Agent) on page 388</td>
<td>Resets the properties of a REST Management Agent to its default values. All the properties of the REST Management Agent are in its default.props file.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>RESTMAN resetstats (Management Agent) on page 391</td>
<td>Resets the statistics of the REST Management Agent.</td>
</tr>
<tr>
<td>RESTMAN setdefault (Management Agent) on page 392</td>
<td>Sets one of the properties of the REST Management Agent to its default value. All the properties of the REST Management Agent are in its <code>default.props</code> file.</td>
</tr>
<tr>
<td>RESTMAN setprops (Management Agent) on page 393</td>
<td>Sets one of the default properties associated with the REST Management Agent.</td>
</tr>
</tbody>
</table>

For more information on the functions of the RESTMAN utility, see Command and Utility Reference on page 357.

### Functions for managing a REST Web application

Other functions of the RESTMAN utility let you manage REST Web applications deployed by the REST Management Agent on the Web server. These are summarized in the following table.

**Table 83: REST Web application functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESTMAN deploy on page 381</td>
<td>Deploys a REST Web application (WAR file) on the Web server. When the application is deployed, its status is disabled. That is, it cannot accept client requests.</td>
</tr>
<tr>
<td>RESTMAN disable on page 381</td>
<td>Makes a deployed REST Web application unavailable to incoming client requests by setting the <code>serviceAvailable</code> property of the REST Web application to 0.</td>
</tr>
<tr>
<td>RESTMAN enable on page 382</td>
<td>Makes a deployed REST Web application available to incoming client requests by setting the <code>serviceAvailable</code> property of the REST Web application to 1.</td>
</tr>
<tr>
<td>RESTMAN getprops (Application) on page 383</td>
<td>Displays all the properties of a REST Web application. All the properties of the Application are in its <code>runtime.props</code> file.</td>
</tr>
<tr>
<td>RESTMAN getstats (Application) on page 384</td>
<td>Displays the statistics for a REST Web application.</td>
</tr>
</tbody>
</table>
| RESTMAN query (Application) on page 387 | Displays the following information about a REST Web application:  
  - Target URI and Statistics URI  
  - Status (whether the application is enabled or disabled)  
  - Relevant application properties (regardless of whether the application is enabled or disabled)  
<p>|</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESTMAN republish (Application) on page 389</strong></td>
<td>Redeploys only the selected files that you specify in the command to their corresponding locations in the REST Web application's folder on the Web server.</td>
</tr>
<tr>
<td><strong>RESTMAN setprops (Application) on page 393</strong></td>
<td>Reinitializes a REST Web application's <code>runtime.props</code> file to the current value of the REST Management Agent's <code>default.props</code> file.</td>
</tr>
<tr>
<td><strong>RESTMAN resetstats (Application) on page 389</strong></td>
<td>Resets the statistics of a REST Web application.</td>
</tr>
<tr>
<td><strong>RESTMAN setprops (Application) on page 393</strong></td>
<td>Sets one of the properties of a REST Web application. All the properties of the Application are in its <code>runtime.props</code> file.</td>
</tr>
<tr>
<td><strong>RESTMAN undeploy on page 394</strong></td>
<td>Undeploys a REST Web application from a REST Management Agent.</td>
</tr>
<tr>
<td><strong>RESTMAN unpublish (Application) on page 394</strong></td>
<td>Deletes a REST service (paar) from a REST Web Application.</td>
</tr>
</tbody>
</table>

For more information on the functions of the RESTMAN utility, see Command and Utility Reference on page 357.
This chapter gives an overview of the security configurations provided by the REST Management Agent, describes the configuration models, and provides information on working with security configurations.

For details, see the following topics:

- Overview
- Security configuration models
- Working with security configurations
- Using LDAP in security configurations
- Using Single Point of Authentication security configurations
- Using SAML in security configurations

Overview

OpenEdge REST Management Agent is a Web application, just as the REST Web application. Therefore, the security configurations for a REST Management Agent and a REST Web application are the same.

OpenEdge supplements the Java container’s security with the industry recognized Spring security framework. Spring security is a customizable authentication and access-control framework. It is one of the industry standards for securing Spring-based applications. The Spring security configurations available to you for securing the RESTful web-services developed in OpenEdge are described in Security configuration models on page 308.
For more information on Spring security framework, see http://docs.spring.io/spring-security/site/docs/4.0.0.RC1/reference/htmlsingle/.

Security configuration models

OpenEdge provides several security configuration models for the OpenEdge REST Management Agent and the REST Web applications. This section describes the default configurations.

For more information on working with security configurations, see Working with security configurations on page 311.

OpenEdge provides the following security configuration models:
### Table 84: REST Security Configurations

<table>
<thead>
<tr>
<th>Configuration Model</th>
<th>OpenEdge Configuration template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>appSecurity-basic-local.xml</td>
<td>This is the <strong>default</strong> configuration model for the OpenEdge REST Management Agent. The Basic configuration authenticates clients using HTTP Basic authentication for user accounts, based on the user information in the <code>users.properties</code> file of the REST Management Agent.</td>
</tr>
<tr>
<td></td>
<td>appSecurity-basic-ldap.xml</td>
<td>This configuration model authenticates clients using HTTP Basic authentication for user accounts, based on the user information stored in the directory governed by LDAP (Lightweight Directory Access Protocol). If you choose this security model, proceed to <a href="#">Using LDAP in security configurations</a> on page 323 to follow the instructions on applying the security configurations.</td>
</tr>
<tr>
<td></td>
<td>appSecurity-basic-oerealm.xml</td>
<td>This configuration model authenticates clients using HTTP Basic authentication for user accounts stored in the OpenEdge Database. If you choose this security model, proceed to <a href="#">Using Single Point of Authentication security configurations</a> on page 324 to follow the instructions on applying the security configurations.</td>
</tr>
<tr>
<td></td>
<td>appSecurity-basic-saml.xml</td>
<td>This configuration model authenticates clients using HTTP Basic authentication for user accounts stored in the database specified by SAML security specifications. If you choose this security model, proceed to <a href="#">Using SAML in security configurations</a> on page 336 to follow the instructions on applying the security configurations.</td>
</tr>
<tr>
<td>Configuration Model</td>
<td>OpenEdge Configuration template</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Form</td>
<td>appSecurity-form-local.xml</td>
<td>This model is very similar to the Basic configuration Model, which authenticates the user accounts based on the user information in the users.properties file of the REST Management Agent. This model authenticates clients using the generic HTTP Form login authentication for user accounts, based on the users.properties file of the OpenEdge REST Management Agent. For more information on managing local user accounts, see the Spring Security framework documentation.</td>
</tr>
<tr>
<td></td>
<td>appSecurity-form-ldap.xml</td>
<td>This model authenticates clients using the generic HTTP Form login authentication for user accounts stored in the directory governed by LDAP (Lightweight Directory Access Protocol). If you choose this security model, proceed to Using LDAP in security configurations on page 323 to follow the instructions on applying the security configurations.</td>
</tr>
<tr>
<td></td>
<td>appSecurity-form-oerealm.xml</td>
<td>This model authenticates clients using the generic HTTP Form login authentication for user accounts stored in the OpenEdge Database. If you choose this security model, proceed to Using Single Point of Authentication security configurations on page 324 to follow the instructions on applying the security configurations.</td>
</tr>
<tr>
<td></td>
<td>appSecurity-form-saml.xml</td>
<td>This configuration model authenticates clients using HTTP Form login authentication for user accounts stored in the database specified by SAML security specifications. If you choose this security model, proceed to Using SAML in security configurations on page 336 to follow the instructions on applying the security configurations.</td>
</tr>
<tr>
<td>Configuration Model</td>
<td>OpenEdge Configuration template</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Anonymous</td>
<td>appSecurity-anonymous.xml</td>
<td>This is the default configuration model for the REST Web applications you develop. With the anonymous security model, you make the Web service public. That is, any user can make requests to the Web service.</td>
</tr>
<tr>
<td>Container</td>
<td>appSecurity-container.xml</td>
<td>This model integrates Spring Security framework with the authentication service of the Java container. The Java container authenticates, and the Spring Security framework controls the authorization to REST Web application resources. For example, if Tomcat is your choice of Web server, you apply security configurations in the tomcat-users.xml (By default, this document is loaded from $CATALINA_BASE/conf/ folder). For more information on container configuration, see the Tomcat documentation.</td>
</tr>
</tbody>
</table>

The OpenEdge security configuration templates are available in the WEB-INF folder of the REST Management Agent or application after you install it and start your Web server.

You choose and apply the security model using the web application's web.xml configuration file. For more information, see Choosing and applying a security configuration on page 312.

For more information on the installation and location of the REST Management Agent, see the Installing and Configuring REST Management Agent on page 261. For instance, if you are using a Tomcat Web server, the configuration templates will be available at <tomcat-installation-directory>/webapps/oerm/WEB-INF.

**Note:** The Spring Security framework enables you to implement other types of authentication systems. For more information on implementing authentication systems, see the Spring Security framework documentation.

---

**Working with security configurations**

This section helps you understand how to work with REST Management Agent and REST Web application security configurations. Administrators must follow the step-by-step process to understand, evaluate, and modify the security configurations:

1. **Choosing and applying a security configuration** on page 312
2. **Adding and modifying a user** on page 313
3. **Setting hashed and salted passwords using Jacksum** on page 315
Choosing and applying a security configuration

Security configurations are defined in the web.xml file of the REST Management Agent or REST Web application. Based on your REST Web service requirements, you choose one of the security configurations models, as described in Security configuration models on page 308.

To choose and apply security configuration:

1. Open the WEB-INF folder of the REST Management Agent or REST Web application in your Web server directory.

Note: For more information on the installation and the location of the REST Management Agent, see Installing and Configuring REST Management Agent on page 261.

2. Open the web.xml file in an XML editor.

3. Find the <context-param> tag. Uncomment one of the <param-value> from the list of security configurations in the <param-value> code block.

For instance, the following <param-value> code block marks Form security as your choice of security configuration model.

```
<param-name>contextConfigLocation</param-name>
<param-value>
  <!--
  /WEB-INF/appSecurity-basic-local.xml
  /WEB-INF/appSecurity-basic-ldap.xml
  /WEB-INF/appSecurity-basic-oerealm.xml
  /WEB-INF/appSecurity-form-ldap.xml
  /WEB-INF/appSecurity-form-oerealm.xml
  /WEB-INF/appSecurity-anonymous.xml
  /WEB-INF/appSecurity-container.xml
  /WEB-INF/appSecurity-form-saml.xml
  /WEB-INF/appSecurity-basic-saml.xml
  -->
  /WEB-INF/appSecurity-form-local.xml
</param-value>
```

Note: If you chose the Container security (/WEB-INF/appSecurity-container.xml) as your security configuration model, you enable the Web application’s use of the Tomcat container’s authenticated user accounts and role definitions.

Note: The Spring Security configurations extends the Web application’s security and uses the container’s authenticated user accounts and assigned roles for authorization. So, if you chose the Container security model, you must edit user accounts information in two configuration files, web.xml and appSecurity-container.xml, for authorization to Web resources. To edit user accounts information in the web.xml file, search for the BEGIN tag (<!-- BEGIN:container.security--> in the web.xml code and follow the instructions given by the NOTE, right below the BEGIN tag, in the code. To edit user accounts information in the appSecurity-container.xml, see Modifying user roles and privileges on page 319.

5. If you chose the LDAP security model, `appSecurity-XXXX-ldap.xml`, you must proceed to Configure the LDAP security configuration model on page 324 to apply security configurations. If you chose the SPA security model, `appSecurity-XXXX-oerealm.xml`, you must proceed to Configure the SPA security configuration model on page 330 to apply security configurations. If you chose the SAML security model, you must proceed to Using SAML in security configurations on page 336. Otherwise, proceed to Adding and modifying a user on page 313.

As per your choice of Security configuration model, you must ensure that you have appropriately set the `webServerAuth` property in the `ubroker.properties` file in OpenEdge (by default, `$DLC\properties\ubroker.properties`). For more information on the values that the `webServerAuth` property can take, see the descriptions in the `ubroker.properties` file.

### Adding and modifying a user

After having chosen a security configuration model for the REST Management Agent or the REST Web application, you can add, remove, modify users and user authentication settings using the `users.properties` file in the `WEB-INF` folder of the REST Management Agent or the REST Web application in your Web server directory.

For more information on the location of the REST Management Agent, see Installing and Configuring REST Management Agent on page 261. For more information choosing and applying security configuration model, see Choosing and applying a security configuration on page 312.

The following sample illustrates the editing operations using the `users.properties` file:

```plaintext
restmgr=password,ROLE_PSCAdmin,ROLE_PSCOper,ROLE_PSCUser,enabled
restoper=password,ROLE_PSCOper,ROLE_PSCUser,enabled
restuser=password,ROLE_PSCUser,enabled
```

After editing user properties in the `users.properties` file:

```plaintext
restmgr=password,ROLE_PSCAdmin,ROLE_PSCUser,ROLE_PSCNewUser,enabled
restoper=password,ROLE_PSCOper,ROLE_PSCUser,enabled
restuser=password,ROLE_PSCUser,ROLE_PSCNewUser,disabled
restNewUserGroup=password,ROLE_PSCNewUser,enabled
```

The following operations were performed:

1. Added "ROLE_PSCNewUser" role to the two users, "restmgr" and "restuser".
2. Removed "ROLE_PSCOper" user from the "restmgr" user.
3. Disabled "restuser" user.
4. Added a new user "restNewUserGroup".

The contents of the `users.properties` file are not secure as they are stored as plain text. The contents in the file are shipped as an example. Typically, a production installation would use the Spring Security configuration for hashed and salted password storage, and then add the secure passwords to the `users.properties` file as illustrated earlier in this section. For more information, see Setting hashed and salted password using EncodePassword on page 313.

### Setting hashed and salted password using EncodePassword

Progress Software recommends that you use the `EncodePassword` utility to generate encrypted passwords. You can add user accounts with hashed and salted passwords in the `users.properties` file.
Note:

EncodePassword is only available for REST Web applications developed in OpenEdge 11.4 or later releases.

If your REST Web application does not have EncodePassword, you must use the open source utility, Jacksum, to generate hashed and salted user passwords. For more information on Jacksum, see Setting hashed and salted passwords using Jacksum on page 315.

For more information on adding user accounts, see Adding and modifying a user on page 313. For more information on hash and salt, refer to the hash and salt cryptography references online.

To use the EncodePassword utility to implement hash and salt user passwords for the REST Web application, myApplication:

1. Execute the following to open the WEB-INF directory of the REST Web application, myApplication:

   ```
   CD \Progress\OpenEdge\servers\tomcat\webapps\myApplication\WEB-INF\n   ```

2. Execute `EncodePassword` located in the REST Web application specifying the user name, `newuser`, which acts as a salt source and the desired password to be encrypted, `newpassword`:

   ```
   EncodePassword -A sha -U newuser -P newpassword
   ```

   You receive the following output:

   ```
   appSecurity sha1 password: Y9+iCjOORqoTWdRgjd/81Ms9w=
   OERM oech1 password: 3e373837333620243f202b
   ```

   The encoded password for the user, `newuser`, is `Y9+iCjOORqoTWdRgjd/81Ms9w=`

   Note:

   This example uses the sha algorithm to hash the `newpassword` string, and the `newuser` string as salt. You can use any of the popular hashing techniques, such as MD5, SHA256 and SHA512. And, you can use any string as salt in the encoding process.

   The `EncodePassword` utility generates another password, `3e373837333620243f202b`, using the OpenEdge encryption algorithm, `oech1`, for encrypting the shared password used between the OpenEdge REST Management Agent and the OpenEdge REST Web applications. For more information on where and how to use and encrypt a password, see Encrypting a password shared by REST Management Agent and REST Web application on page 317.

3. Open the security configuration file you chose in Choosing and applying a security configuration on page 312 and uncomment the code snippet that implements password hashing in REST Web application.

   For example, assume that you are using the Basic security configuration, `appSecurity-basic-local.xml`. Do the following:
a) Navigate to and uncomment the code snippet in bold in `appSecurity-basic-local.xml`:

```xml
<authentication-manager id="RestApplicationtAuth"
    <authentication-provider>
        <!-- Uncomment to add strong password hashing in
            users.properties
            <password-encoder hash="sha" base64="true">
                <salt-source user-property="username" />
            </password-encoder>
        -->
        <user-service properties="/WEB-INF/users.properties" />
    </authentication-provider>
</authentication-manager>
```

After uncommenting, the code snippet must look as follows:

```xml
<authentication-manager id="RestApplicationtAuth"
    <authentication-provider>
        <password-encoder hash="sha" base64="true">
            <salt-source user-property="username"/>
        </password-encoder>
        <user-service properties="/WEB-INF/users.properties"/>
    </authentication-provider>
</authentication-manager>
```

b) Save the security configuration file.

4. Add the new user, newuser, and the encoded password you generated in 2 on page 314, to the `users.properties` file by appending the following line of code to it:

```
newuser=Y9+iCjOORQoqoTWdRgjd/81Ms9w=,ROLE_PSCAdmin,enabled
```

In this example, you assigned the existing `ROLE_PSCAdmin` role to the user, newuser, and enabled newuser to work with the REST Management Agent.

**Note:** The `users.properties` file cannot store plain text and encoded passwords. Therefore, before adding a new user with an encoded password, you must ensure that all the other users have encoded passwords.

5. Save and close the `users.properties` file.

**Setting hashed and salted passwords using Jacksum**

Progress Software recommends that you use the EncodePassword utility to generate hashed and salted passwords. For more information on the EncodePassord utility, see Setting hashed and salted password using EncodePassword on page 313.

However, as an alternative to the EncodePassword utility, you use the Open Source software, Jacksum, to generate the encrypted password. You can then add the user accounts in the `users.properties` file with hashed and salted passwords.
For more information on adding user accounts, see Adding and modifying a user on page 313. For more information on hashing and salt, refer to the Hashing and Salt cryptography references online.

To use Jacksum to add a new user account, in the users.properties file, with an hashed and salted password:

1. Download and install the Jacksum software by following the instructions in http://www.jonelo.de/java/jacksum/#Installation.

2. Execute the following command to open the Jacksum installation directory:

   ```
   CD <installation-dir>\jacksum <version>
   ```

3. Execute the following command to get an encoded password for the required string. In this example, you are encoding the string `newpassword{newuser}`:

   ```
   java -jar jacksum.jar -a sha -E base64 -q "txt:newpassword{newuser}"
   ```

   In this example, you encoded the string `newpassword{newuser}`, and Jacksum returned the encoded password, `Y9+iCjOORQoqoTWdRgjd/81Ms9w=`

   **Note:**  
   This example uses the *sha* algorithm to hash the `newpassword` string, and the `newuser` string in the curly braces as salt.

   You can use any of the popular hashing techniques that Jacksum supports, such as *CRC*, *SHA*, *Tiger*, *sum* and *Whirlpool*. And, you can use any string as salt in the encoding process.

4. Open the Security configuration file you choose in Choosing and applying a security configuration on page 312 and uncomment the code snippet that implements password hashing in the REST Web application.

   In this example, assume that you are using the Basic security configuration, `appSecurity-basic-local.xml`. Do the following:

   a) Navigate to and uncomment the code snippet in bold in the `appSecurity-basic-local.xml`:

   ```
   <!-- Uncomment to add strong password hashing in users.properties
   <authentication-manager id="RestApplicationtAuth"
   <authentication-provider>
   <!-- Uncomment to add strong password hashing in
   users.properties
   <password-encoder hash="sha" base64="true">
   <salt-source user-property="username" />
   </password-encoder>
   
   -->
   <user-service properties="/WEB-INF/users.properties" />
   </authentication-provider>
   </authentication-manager>
   ```
After uncommenting, the code snippet must look as follows:

```xml
<authentication-manager id="RestApplicationtAuth"
  <authentication-provider>
    <password-encoder hash="sha" base64="true">
      <salt-source user-property="username" />
    </password-encoder>
    <user-service properties="/WEB-INF/users.properties" />
  </authentication-provider>
</authentication-manager>
```

b) Save the security configuration file, appSecurity-basic-local.xml.

5. Add the new user account, newuser, and the encoded password (that you generated in 3 on page 316) to the `users.properties` file by appending the following line of code in it:

```
newuser=Y9+iCjOORQoqoTwdRgjd/81Ms9w=,ROLE_PSCAdmin,enabled
```

In this example, you assigned the existing `ROLE_PSCAdmin` role to the `newuser` and enabled the user account `newuser` to work with the REST Management Agent.

**Note:** The `users.properties` file cannot store both plain text and encoded passwords. Therefore, before adding a new user with an encoded password, you must ensure that all the existing users and the users you add to the `users.properties` file have encoded passwords.

6. Save and close the `users.properties` file.

### Encrypting a password shared by REST Management Agent and REST Web application

The REST Management Agent uses the `/adapterman` URI to communicate with REST Web applications. The `/adapterman` URI is to be used only in OpenEdge as a cross-context communication port. For security, the `/adapterman` URI has its own security configuration that comprises a shared password used between the OpenEdge REST Management Agent and OpenEdge REST Web applications.

In a production environment, the `/adapterman` URI is secured with the shared password. The password does not have to be modified as it is consumed by the OpenEdge REST Management Agent and OpenEdge REST Web application. However, as part of advanced security measures for the REST Web application, you can encrypt the shared password.

To encrypt the shared password, do the following:

1. Generate a hashed and salted password for a REST Web application user using the `EncodePassword` utility. The utility provides two passwords, one for the REST Web application user, and another for the REST Management Agent.

   For more information on generating hashed and salted passwords, see the Setting hashed and salted password using `EncodePassword` on page 313.
For example, if the EncodePassword utility gives the following output when generating a password for a user, newuser, where the user name also acts as a salt source:

```
appSecurity sha256 password:tPcPFxLM11lk0AdT75Jol1ooXl6Rw8UtSP3LhmhUQVg= ORM oech1 password: 3e373837333620243f202b
```

The password, `3e373837333620243f202b`, generated using the OpenEdge encryption algorithm, oech1, is required for encrypting the shared password used between the OpenEdge REST Management Agent and the OpenEdge REST Web applications.

2. Open the `web.xml` file in the REST Management Agent in a text editor, look for the following code snippet:

```
<context-param>
  <param-name>adaptermanID</param-name>
  <param-value>C62384a0F1516B00</param-value>
</context-param>
<context-param>
  <param-name>adaptermanSecret</param-name>
  <param-value>1f177e7600202027731f2e293322362174</param-value>
</context-param>
```

Replace the `param-value` of `adaptermanID` with the new user name and `adaptermanSecret` with `new oech1 encoded password`. Extending the example discussed in 1 on page 317, you must make changes, as highlighted:

```
<context-param>
  <param-name>adaptermanID</param-name>
  <param-value>newuser</param-value>
</context-param>
<context-param>
  <param-name>adaptermanSecret</param-name>
  <param-value>3e373837333620243f202b</param-value>
</context-param>
```

3. Open the security configuration file, `appSecurity-xxxx.xml`, of your REST Web application and uncomment the code snippet to implement password hashing in it.

Extending the example discussed in 1 on page 317, if you are using the security configuration, `appSecurity-anonymous.xml`, for your REST Web application.

Do the following
a) Find the `<authentication-manager id="RestManagerAuth" >` code snippet and make the updates highlighted to reflect the information about the encryption algorithm employed (in this case, SHA), and the newly generated user name (in this case, newuser) and password (in this case, newuser):

```xml
<!-- REST Manager access --
<authentication-manager id="RestManagerAuth" >
    <authentication-provider>
        <password-encoder hash="sha-256" base64="true">
            <salt-source user-property="username" />
        </password-encoder>
        <user-service>
            <user name="newuser"
                  password="tPcPFxLM11LkOAdT7SJol1ocXl6Rw8UtSP3LhmhUQVg="
                  authorities="ROLE_PSCAdapter" />
        </user-service>
    </authentication-provider>
</authentication-manager>
```

b) Save the security configuration file.

4. Restart the Web server, and then re-enable the deployed REST Web application.

**Modifying user roles and privileges**

For all the existing users in the `users.properties` file, you can modify the user roles and privileges using the appropriate security configuration file, `appSecurity-XXXX.xml`, in the `WEB-INF` folder of the REST Management Agent or the REST Web application in your web server directory. Irrespective of which security configuration model you chose, you perform a similar set of edits.

For example, the following assigns privileges to `ROLE_PSCAdmin` and `ROLE_PSCOper` in the `appSecurity-form-local.xml` file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Get Manager status -->
    <intercept-url pattern="/status" method="GET"
                  access="hasAnyRole('ROLE_PSCAdmin','ROLE_PSCOper')"/>
```

Add the user role, `ROLE_PSCNewUser`, with privileges of performing GET operation on `/status`:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Get Manager status -->
    <intercept-url pattern="/status" method="GET"
                  access="hasAnyRole('ROLE_PSCAdmin','ROLE_PSCOper','ROLE_PSCNewUser')"/>
```
Extending CORS support

Cross-origin resource sharing (CORS) is a W3C group standard that allows an HTTP client to access a Web page loaded from one domain, to make a JavaScript, XMLHttpRequests, to access resources located in another domain. Such “cross-domain” requests are otherwise forbidden by the Web browser's Java Script security default security policy. The CORS standard defines a way in which a client can ask a server if it can access a cross-origin request. The server's configuration helps in determining whether the cross-domain request is granted.

The W3C CORS standard works by adding new HTTP headers that allow servers to permitted domains. Browsers support these headers and enforce the restrictions these headers establish. Additionally, for HTTP request methods that might affect user data (in particular, HTTP methods other than the GET method, or POST method with certain MIME types), the specification mandates that the browser preflight the request, soliciting supported methods from the server with an HTTP OPTIONS request header, and then, upon approval from the server, send the actual request with the actual HTTP request. Servers can also notify clients whether user credentials (including cookies and HTTP authentication data) be sent with requests.

Although most modern browsers support CORS, you should verify CORS support in your browser.

A CORS enabled server or web application classifies all HTTP requests as:

- A CORS request that contains an HTTP Origin header
- A preflight request that contains the Access-Control-Request-Method header in an OPTIONS request
- A generic request that does not contain any CORS HTTP headers

For more information on CORS standard and the advances in the standard, see the documentation at http://www.w3.org/TR/cors/.

**Note:** CORS can be used as an alternative to JSON with padding (JSONP) pattern. While JSONP supports only the GET request method, CORS also supports other types of HTTP requests. Using CORS enables a Web programmer to use regular XMLHttpRequests, which supports better error handling than JSONP. Also, while JSONP can cause XSS issues where the external site is compromised, CORS allows web sites to manually parse responses to ensure security.

OpenEdge uses a third-party Java open source package, CORS Filter, in which all of the primary CORS functionality resides. OpenEdge has integrated CORS filter into the Java container Web applications by implementing a Spring Security filter bean so that the CORS filter can be configured from within the Spring Security configuration files, with all of the other Web application security.

Therefore, the CORS support is already incorporated into the REST Web application security policy, and is enabled for all generic requests and CORS requests from any domain.

Customizing CORS support

To customize CORS support:

1. Identify and open the security configuration you applied to your REST Web application.

   Your REST Web application's security configuration is based on your performing the instructions in Choosing and applying a security configuration on page 312.

2. In the security configuration file, appSecurity-XXX.xml, uncomment only the required properties in the OECORSFilter bean code snippet to enable CORS support.
Note:

Typically, you uncomment the allowAll property and set it to false to ensure that all HTTP clients make CORS requests by sending an Origin header.

Uncomment the allowDomains property to limit CORS access to only certain HTTP clients. And, uncomment and use messageHeaders and responseHeaders properties if the HTTP clients require sending/receiving headers not contained in the defaults.

You typically uncomment only the allowAll and allowDomains properties to customize CORS support. Progress Software recommends that you uncomment only the properties you need to set; otherwise, let the default settings be applied to the properties.

If you uncomment all the CORS properties, the OECORSFilter bean code snippet looks similar to the following:

```xml
<bean id="OECORSFilter" class="com.progress.rest.security.OECORSFilter">
  <property name="allowAll" value="false" />
  <property name="allowDomains" value="*" />
  <property name="allowSubdomains" value="false" />
  <property name="allowMethods" value="GET,PUT,POST,DELETE" />
  <property name="responseHeaders" value="Cache-Control, Content-Language, Content-Type, Expires, X-CLIENT-CONTEXT-ID" />
  <property name="supportCredentials" value="true" />
  <property name="maxAge" value="-1" />
</bean>
```

3. After uncommenting the required properties from the OECORSFilter bean code snippet, for each of the properties, such as allowAll and allowDomains, you must update the value attribute. For information about the OECORSFilter properties, see OECORSFilter properties on page 321.

4. Save the security configuration file, appSecurity-XXX.xml.

Note: You must restart the Web server for the above security configuration updates to take effect.

OECORSFilter properties

The OECORSFilter is a standard Spring security filter bean. It is declared in the Spring security configuration files (appSecurity-XXX.xml) in which its properties can be customized to override the defaults.

Progress Software recommends that you uncomment only the properties you need to set. The following table describes all the property names:
Table 85: OECORSFilter properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
<th>Data types</th>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowAll</td>
<td>Specifies that CORS filter allow every client request. If this property is set</td>
<td>Boolean</td>
<td>true</td>
<td>true or false</td>
</tr>
<tr>
<td></td>
<td>to true, all the other CORS properties values are ignored by the CORS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>filter.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allowDomains</td>
<td>Specifies the domains that can make server requests.</td>
<td>String</td>
<td>*</td>
<td>{&quot;*&quot;, &quot;domain1&quot;, &quot;domain2&quot;} }</td>
</tr>
<tr>
<td>allowSubdomains</td>
<td>Specifies if subdomains of the permitted domains be allowed to make server</td>
<td>Boolean</td>
<td>false</td>
<td>true or false</td>
</tr>
<tr>
<td></td>
<td>requests.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allowMethods</td>
<td>Specifies valid HTTP method names.</td>
<td>String</td>
<td>GET, PUT, POST, DELETE</td>
<td>Valid HTTP methods in upper case.</td>
</tr>
<tr>
<td>messageHeaders</td>
<td>Specifies the message header to be passed as a header to the server. If</td>
<td>String</td>
<td>Refer to the footnote.</td>
<td>Any valid string</td>
</tr>
<tr>
<td></td>
<td>you are passing multiple messages, you must specify a comma-separated list of messages.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11 To begin active CORS access control for all clients, turn allowAll property to false. If this property is set to true, the CORS filter grants access to all generic clients, that is, all generic requests (without CORS headers) are allowed to access the resources when the server is using non-standard HTTP or HTTPS ports.

12 Ensure that you supply a fully qualified domain name. You must not append the path (/) separator in the domain name, and you must add a port (#) because the allowDomains default value is *, that is, all client requests are allowed to access the resources.

Ensure that you specify the list of domains in the allowDomains property for the filter to restrict all client requests from domains other than the domains listed in the allowDomains property.

13 This list of method names is for all resources in the web application and needs to be coordinated with the resource authorization controls.

14 This is for advanced HTTP client and server use. Use this only when the web application's clients are coded to use these headers.

By default, the messageHeaders has the following methods: Accept, Accept-Language, Content-Language, Content-Type, X-CLIENT-CONTEXT-ID, Origin,Pragma, Cache-Control, Access-Control-Request-Headers, Access-Control-Request-Method.
Using LDAP in security configurations

This section helps you understand how to use Lightweight Directory Access Protocol (LDAP) for setting up your REST Web application's security configurations.

It assumes that you understand LDAP implementation and user account management, and that you are ready to make use of the distributed directory information services for OpenEdge REST Web application authentication. For any LDAP related information, refer to LDAP administration documentation online.

To use LDAP for your REST Web application's security configurations:

1. Choose and apply an LDAP security configuration on page 324
2. Configure the LDAP security configuration model on page 324

For more information on using LDAP security configurations, see Choose and apply an LDAP security configuration on page 324 and Configure the LDAP security configuration model on page 324.

---

15 This is for advanced HTTP client and server use. Use this only when the web application's clients are coded to use these headers.

By default, the responseHeaders has the following methods: Cache-Control, Content-Language, Content-Type, Expires, X-CLIENT-CONTEXT-ID.

16 The default is true, which allows the client to use user logic sessions via COOKIES. If you do not want the client to send COOKIE user credentials, set the property value to false.

By setting the property to false, you supply totally stateless resources to anonymous users.
Choose and apply an LDAP security configuration

Choose one of the LDAP security configurations from the available security configurations described in Security configuration models on page 308, and then apply that LDAP security configuration model by following the instructions in Choosing and applying a security configuration on page 312.

Configure the LDAP security configuration model

After choosing the LDAP security configuration model in the in Choose and apply an LDAP security configuration on page 324, you must configure the LDAP security model, appSecurity-XXXX-ldap.xml, as instructed in the following table:

Table 86: LDAP security configuration settings

<table>
<thead>
<tr>
<th>Find</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;ldap-server id=&quot;PrimayLDAP&quot; url=&quot;ldap://enter_hostname:389/root_dn_here&quot; manager-dn=&quot;uid=admin,ou=system&quot; manager-password=&quot;admin&quot; /&gt;</code></td>
<td>The LDAP id, url, manager-dn (Account manager domain name), and manager-password (Account manager's password) fields for your REST Web application to identify and authenticate your LDAP server accounts.</td>
</tr>
<tr>
<td><code>&lt;authentication-manager id=&quot;RestApplicationtAuth&quot; &gt;</code> &lt;ldap-authentication-provider server-ref=&quot;PrimayLDAP&quot; group-role-attribute=&quot;cn&quot; role-prefix=&quot;ROLE_&quot; group-search-filter=&quot;{(member={0})}&quot; group-search-base=&quot;&quot; user-search-base=&quot;&quot; user-search-filter=&quot;{(uid={0})}&quot; /&gt; `authentication-manager&gt;</td>
<td>All the properties of the LDAP authentication manager based on the design and implementation of your directory services.</td>
</tr>
</tbody>
</table>

After using LDAP for your REST Web application, you can also extend CORS support using the instructions provided in Extending CORS support on page 320.

Using Single Point of Authentication security configurations

This section helps you understand how to use the OpenEdge Single Point of Authentication (SPA) configurations to configure your REST Web application's security.

SPA in OpenEdge enables you to reuse an AppServer as the source of user account information for the authentication process. This reusability is provided by the Spring Security framework. Spring Security requires a separate authentication provider plug-in for each user account information source, such as an RDBMS database or a local text file. The Spring Security authentication process uses an object model that performs certain tasks:

- **Authentication manager**: The manager is responsible for creating and calling one of more authentication providers until the authentication process is complete (success or failure).
- **Authentication provider**: The provider is called by the authentication manager and is responsible for creating the user details service.
• **User details service:** This service is created by the authentication provider and is responsible for interacting with the account system to retrieve the user account along with the account attributes information, such as password and user roles. This information is used for validating the account password and for creating a Spring Security authentication token.

• **Authentication token:** This token is created as the result of a successful Spring Security authentication process. The token contains authenticated user information, such as user ID, granted roles, and session ID.

---

*Note:* The authentication token is used to generate a Spring ClientPrincipal token that can be used further in the authorization process.

---

The OpenEdge SPA implementation extends the Spring Security authentication process. The implementation consists of 2 components:

• **OpenEdge Realm (OERealm) Client:** The client contains the Spring Security authentication provider and user account service objects that interact with the AppServer to retrieve the user account information. This information is then sent to the Spring Security authentication process.

  The OERealm Client does the following:
  1. Connects to the AppServer.
  2. Executes a request to locate the user account and retrieve the account unique numeric ID.
  3. Executes requests to retrieve the user account details in the following order:
     a. Granted roles
     b. The account enabled state
     c. The account locked state
     d. The account expired state

  *Note:* If the OERealm client finds that the account does not exist or if the account is in the disabled, locked, or expired state, no additional AppServer requests are performed.

---

• **OERealm AppServer ABL interface:** The service interface class that implements the OERealm ABL interface must also implement the OERealm component on the AppServer.

  The actions of the class are performed in the following order:
  1. Search for the user account using the name (*userid[@domain]*)
  2. Return the user account attributes for the user account.
  3. Validate the user account’s password.

     Optionally, authorize the OERealm client and enable it to receive user account information.

---

For details on how to create an OERealm AppServer ABL Interface see the [OERealm](#) on page 425 section.

To use SPA for configuring your REST Web application's security:

1. Configure an OERealm service on an AppServer on page 326
2. Select and apply an SPA security configuration on page 330
3. Configure the SPA security configuration model on page 330
4. Test the OERealm service on page 335
Configure an OERealm service on an AppServer

This section helps you create an OERealm service interface that helps you find user account information, obtain user attributes, and validate a user account's password.

To configure an OERealm service on an AppServer:

1. Configure a State-free AppServer that is required to host the OERealm service interface class. If your application is not running on a State-free AppServer, you must define and run a State-free AppServer to support the OERealm service interface.

2. Create a singleton OOABL class that implements the OERealm IHybridRealm interface. If you have the OpenEdge Documentation and Samples installed, you can find a sample interface implementation class, HybridRealm.cls, at $DLC\src\samples\security\OpenEdge\security\Realm.

In the sample HybridRealm.cls file, the following methods are declared and defined:

Table 87: Methods defined in the HybridRealm.cls class

<table>
<thead>
<tr>
<th>Method</th>
<th>Mandatory?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValidateUser</td>
<td>Yes</td>
<td>This method searches for the user account, validates whether the account domain is valid and enabled, and returns the account numeric ID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The OERealm service interface uses a numerical user account ID as an ABL integer instead of the full user account name. If the user account storage of your OERealm service does not support the integer user account IDs, it is recommended that you build a mechanism to permanently assign unique ABL integer values to each user account in the storage.</td>
</tr>
<tr>
<td>ValidatePassword</td>
<td>Yes</td>
<td>This method validates the client-supplied password of the user account against the stored password.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> You can define both the clear-text and digest form of the ValidatePassword() method.</td>
</tr>
<tr>
<td>GetAttribute</td>
<td>Yes</td>
<td>This method retrieves an attribute for the specified user account.</td>
</tr>
<tr>
<td>GetAttributeNames</td>
<td>No</td>
<td>This method retrieves the name assigned to the attributes.</td>
</tr>
<tr>
<td>GetUserNames</td>
<td>No</td>
<td>This method retrieves all the user names associated with the user account.</td>
</tr>
<tr>
<td>GetUserNamesByQuery</td>
<td>No</td>
<td>This method retrieves the user names that match the input query string.</td>
</tr>
<tr>
<td>RemoveAttribute</td>
<td>No</td>
<td>This method deletes the value of the given attribute.</td>
</tr>
<tr>
<td>SetAttribute</td>
<td>No</td>
<td>This method assigns a new value to the given attribute.</td>
</tr>
</tbody>
</table>
You can implement the `GetAttribute()` method to return the `UNKNOWN` or `CHARACTER` value. If you have configured the REST adapter with an SPA security configuration model, the `OERealmUserDetailsImpl` module of the OERealm client handles the `UNKNOWN` value by replacing the value with a static, non-configurable default value. If a `CHARACTER` value is returned, the OERealm service interface must encode the value in a format that the `OERealmUserDetailsImpl` module can understand. The known values of the `OERealmUserDetailsImpl` module are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Character encoding format</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_ROLES</td>
<td>string-extent</td>
<td>&quot;&quot; or &quot;xxxx[,yyyy,zzzz]&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The OERealm service interface must return roles</td>
</tr>
<tr>
<td></td>
<td></td>
<td>without the <code>ROLE_</code> prefix. For example, the interface must</td>
</tr>
<tr>
<td></td>
<td></td>
<td>return PSCUser instead of ROLE_PSCUser.</td>
</tr>
<tr>
<td>ATTR_ENABLED</td>
<td>boolean</td>
<td>A character from the {T,t,F,f,Y,y,N,n,0,1} set</td>
</tr>
<tr>
<td>ATTR_LOCKED</td>
<td>boolean</td>
<td>A character from the {T,t,F,f,Y,y,N,n,0,1} set</td>
</tr>
<tr>
<td>ATTR_EXPIRED</td>
<td>boolean</td>
<td>A character from the {T,t,F,f,Y,y,N,n,0,1} set</td>
</tr>
</tbody>
</table>

3. Build and test your OERealm service interface. You can perform this step using an ABL client by writing an AppServer test harness that calls the OERealm service interface.

4. Install the REST Web application on the Apache Tomcat Java container.

**Note:** Additional security considerations for the SPA security configuration are discussed in Security considerations for SPA security configurations on page 327.

### Security considerations for SPA security configurations

The SPA security configuration process is used to authenticate users. The process itself must be secure against unwarranted attacks.

The following considerations must be followed for security purposes:

- **OERealm service interface security**

  The OERealm user details service can be configured to send a sealed ClientPrincipal to the AppServer OERealm service interface on each method invoked by the user details service. The OERealm service interface can wrongly intercept the ClientPrincipal from `SESSION:CURRENT-REQUEST-INFO:GET-CLIENT-PRINCIPAL` and validate the client before performing any OERealm operation. Another security flaw occurs when you use the `SECURITY-POLICY:SET-CLIENT()` or `SET-DB-CLIENT()` method for performing OERealm client identity checks. Using these methods changes the current identity of the ABL session, which must then be reestablished after the OERealm operation is complete.
To overcome this problem, it is recommended that you use the `CLIENT-PRINCIPAL:VALIDATE-SEAL()` method along with encoded domain access codes ("oechl::<xxxx>") to validate the ClientPrincipal. You can apply this validation as an operation at the beginning of each OERealm service interface method. This isolates the OERealm use case from the other ClientPrincipal-related activities, such as a session, database, AppServer connection, or BPM identity.

• **OERealmAuthProvider security**

You can configure the OpenEdge domain access code to seal the ClientPrincipal in the Single sign-on (SSO) mode. By default, the Spring Security OERealmAuthProvider bean creates the ClientPrincipal as a single-tenant. The problem is that every OE REST Web application user of the ClientPrincipal is a member of the same OpenEdge domain.

You can enhance OERealmAuthProvider security in one of the following ways:

• **Set the userDomain and key attributes**

You can enhance security by setting the following OERealmAuthProvider attributes while sealing the ClientPrincipal:

• The `userDomain` attribute to set the static OpenEdge domain.

• The `key` attribute to set the domain access code.

These attributes are used to uniquely identify application users based on the user domain. You can also disable access to the OE REST Web application just by disabling the domain. The following table defines the available combinations of domain, domain access code, and user ID:

<table>
<thead>
<tr>
<th>Domain name</th>
<th>Key attribute</th>
<th>User ID</th>
<th>Signing operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot; (default)</td>
<td>&quot;&quot; (default)</td>
<td>&quot;uid&quot; (&quot;uid@&quot;)</td>
<td>Seal the blank domain using the built-in Domain Registry blank access code.</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>&quot;&lt;xxxx&gt;&quot;</td>
<td>&quot;uid&quot; (&quot;uid@&quot;)</td>
<td>Seal the blank domain using the clear text sting value, &quot;&lt;xxxx&gt;&quot;.</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>&quot;oechl::&lt;xxxx&gt;&quot;</td>
<td>&quot;uid&quot; (&quot;uid@&quot;)</td>
<td>Seal the blank domain using the encoded value, &lt;xxxx&gt;.</td>
</tr>
<tr>
<td>&quot;abc&quot;</td>
<td>&quot;&quot; (default)</td>
<td>&quot;uid@abc&quot;</td>
<td>Seal the abc domain using the built-in Domain Registry blank access code.</td>
</tr>
<tr>
<td>&quot;abc&quot;</td>
<td>&quot;&lt;xxxx&gt;&quot;</td>
<td>&quot;uid@abc&quot;</td>
<td>Seal the abc domain using the clear text sting value, &quot;&lt;xxxx&gt;&quot;.</td>
</tr>
<tr>
<td>&quot;abc&quot;</td>
<td>&quot;oechl::&lt;xxxx&gt;&quot;</td>
<td>&quot;uid@abc&quot;</td>
<td>Seal the abc domain using the encoded value, &lt;xxxx&gt;.</td>
</tr>
</tbody>
</table>

• **Set the multiTenant property**

If you enable the `multiTenant` attribute of the OERealmAuthProvider, all the user IDs are appended with the `userDomain` attribute value. In this case, the same key can used to seal the ClientPrincipal for all OE domains.

• **Set the external domain registry for multi-tenant applications**
The OERealmAuthProvider provides support for using an administrator-generated Domain Registry file to sign and seal ClientPrincipals in multi-tenant applications. This file contains a list of domains that are allowed to have access to the ClientPrincipal.

To use the Domain Registry file:

1. Use the gendomreg command, located at $DLC/bin, that takes a formatted text file as input and creates a secure binary Domain Registry file. The text file lists OE domains that are allowed to have access to the ClientPrincipal in the Domain-name=Domain-access-code format. Each domain must be listed in a separate line.

2. Copy the Domain Registry file to the OE REST Web application's /WEB-INF/classes directory.

3. Configure the OERealmAuthProvider bean to load and sign a ClientPrincipal.

• Preventing authorization before authentication

The ClientPrincipal that is generated by OE REST Web application's OERealmAuthProvider supercedes the one that is generated by the OEClientPrincipalFilter for SSO. This is a problem when the OEClientPrincipalFilter authorizes the incorrect ClientPrincipal without any processing. You can disable the OEClientPrincipalFilter by setting its enableCP property to false.

Another potential security problem occurs if a client is able to call the OERealm service interface without authentication. You can secure the service interface against such unwarranted client access by using the ClientPrincipal object and validation technique that is similar to the one that is used in the ABL business logic of your AppServer. To use this validation technique, the OE REST Web application's OERealmUserDetails acts as an AppServer client and identifies itself to the AppServer's OERealm service interface. The service interface validates the client using a ClientPrincipal just like it is used in other parts of the ABL application.

To develop the OE REST Web application's OERealmUserDetails for acting as an AppServer client:

1. Use the genspacp utility, at the $DLC/bin location, to create a sealed ClientPrincipal that represents a valid AppServer client. The utility requires the user ID, domain name, domain access code, and output file name to create a binary file that contains the sealed ClientPrincipal in the <file-name>.cp format. The domain name and domain access code correspond to those that are then used in the OERealm service interface to validate clients.

2. Copy the output file from the $DLC/bin location to the OE REST Web application's WEB-INF/classes directory.

3. Set the realmTokenFile configuration property of OERealmUserDetails to the output file name.

4. In the OERealm service interface, add the ClientPrincipal validation code. The code can reside in the AppServer activate procedure or as step in each of the OERealm class methods. It is recommended that you use the CLIENT-PRINCIPAL:VALIDATE-SEAL("domain-access-code") method for validation. You must ensure that domain-access-code is in the encoded "oech1::xxxx" format.

When the OERealm service interface validates the ClientPrincipal in the created binary output file, the following sequence occurs:

1. Once the OE REST Web application is loaded, if the realmTokenFile configuration property of OERealmUserDetails contains the output file name, the file is used to load the ClientPrincipal. If the output file does not load, an error is added to the error log.

2. For each user authentication request, OERealmUserDetails is used to send the ClientPrincipal to the AppServer OERealm service interface methods.

3. The OERealm service interface validates the ClientPrincipal. The following validation outcomes are possible:
   • If the ClientPrincipal is not sent to the interface, an error is returned.
If the ClientPrincipal is sent to the interface, the interface is used to validate the domain access code. If the validation fails, an error is returned.

If the ClientPrincipal validation is successful, the execution of the ABL class method continues.

Select and apply an SPA security configuration

Select one of the SPA security configurations models described in Security configuration models on page 308 (appSecurity-xxxx-oerealm.xml) and then apply the model by following the instructions in Choosing and applying a security configuration on page 312.

Configure the SPA security configuration model

The OpenEdge SPA implementation supplies an authentication provider and a user details services module that accesses an OpenEdge AppServer to obtain user account information and supply it to the Spring security's authentication process. You require the OpenEdge AppServer used by the user details services for installing and supporting an OpenEdge-defined Realm service interface that helps you find user account information, obtain user attributes, and validate the user account's password.

After choosing the SPA security configuration model from Select and apply an SPA security configuration on page 330, you must configure the SPA security model, appSecurity-XXXX-oerealm.xml, as instructed in the following table:

<table>
<thead>
<tr>
<th>Table 90: SPA security configuration settings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Find</strong></td>
</tr>
<tr>
<td>&lt;b:bean id=&quot;OERealmAuthProvider&quot; class=&quot;com.progress.rest.security.OERealmAuthProvider&quot; &gt;</td>
</tr>
<tr>
<td>&lt;b:bean id=&quot;OERealmUserDetails&quot; class=&quot;com.progress.rest.security.OERealmUserDetailsImpl&quot; &gt;</td>
</tr>
</tbody>
</table>

OERealmAuthProvider properties

OERealmAuthProvider is a bean that defines how to create a Spring Authentication token (ClientPrincipal or UserPasswordAuthenticationToken) after an OpenEdge database user account's authentication process is successful.
The ClientPrincipal generated by the OERealmAuthProvider supercedes the ClientPrincipal created by OEOClientPrincipalFilter. Therefore, when you use an SPA security configuration for your REST application, you can optionally disable the OEOClientPrincipalFilter by setting its enableCP property to false. For more information on updating the OEOClientPrincipalFilter, see OEOClientPrincipal properties on page 290.

Set the properties of OERealmAuthProvider per the following table:
Table 91: OERealmAuthProvider properties

<table>
<thead>
<tr>
<th>Spring property</th>
<th>Default value</th>
<th>Range of accepted values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createCPAuthn</td>
<td>&quot;true&quot;</td>
<td>&quot;true&quot;</td>
<td>&quot;false&quot;</td>
</tr>
<tr>
<td>sealClientPrincipal</td>
<td>false</td>
<td>&quot;true&quot;</td>
<td>&quot;false&quot;</td>
</tr>
<tr>
<td>key</td>
<td>&quot;&quot;</td>
<td>&quot;&lt;Valid string&gt;&quot;</td>
<td>Specifies the OpenEdge domain's access code to seal the ClientPrincipal token with the specified string.</td>
</tr>
<tr>
<td>&quot;registryFile&quot;</td>
<td>null</td>
<td>&quot;&lt;valid-file-path&gt;&quot;</td>
<td>The registry file used to seal the ClientPrincipal. It is generated using the gendomreg utility.</td>
</tr>
<tr>
<td>userDomain</td>
<td>&quot;&quot;</td>
<td>&quot;&lt;Valid string&gt;&quot;</td>
<td>Specifies the OpenEdge domain name to append to the user account name if the name does not already contain a user domain.</td>
</tr>
<tr>
<td>Spring property</td>
<td>Default value</td>
<td>Range of accepted values</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------</td>
<td>--------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>multiTenant</td>
<td>&quot;false&quot;</td>
<td>&quot;true&quot;</td>
<td>Prescribes the user to enter a fully qualified OpenEdge user-id@domain tenant when set to true.</td>
</tr>
<tr>
<td>authz</td>
<td>&quot;true&quot;</td>
<td>&quot;true&quot;</td>
<td>Loads the user account's roles as Spring authorities used in the Spring security authorization process.</td>
</tr>
<tr>
<td>properties</td>
<td>&quot;empty&quot;</td>
<td>&quot;valid name and value&quot;</td>
<td>Loads static client-principal properties by name and value.</td>
</tr>
<tr>
<td>expires</td>
<td>&quot;0&quot;</td>
<td>&quot;0&quot; or &quot;positive integer&quot;</td>
<td>Expires ClientPrincipal in the specified number of seconds.</td>
</tr>
</tbody>
</table>

**OERealmUserDetails properties**

OERealmUserDetails is a bean that you configure to set how the authenticated OpenEdge users connect to a state-free AppServer. By configuring this bean, you describe how a user account's details must be passed between Spring Security framework and the OpenEdge AppServer.

Set the properties of OERealmUserDetails using the following information:

---

17 The key property can also be in the "oecli::<hex-string>" where <hex-string> is the output from the OE genpassword utility.
Table 92: OERealmUserDetails properties

<table>
<thead>
<tr>
<th>Spring property</th>
<th>Default value</th>
<th>Range of accepted values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>realmURL</td>
<td>&quot;Appserver://localhost:5162/oerealm&quot;</td>
<td>&quot;valid URL&quot;</td>
<td>Specifies the URL to connect users to the state-free AppServer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Note:</strong> <em>oerealm</em> is the name of the OpenEdge State-free AppServer where you deploy the OpenEdge realm. Refer to <a href="#">Configure an OERealm service on an AppServer</a> on page 326.</td>
</tr>
<tr>
<td>realmClass</td>
<td>&quot;OpenEdge.Security.Realm.IHybridRealm&quot;</td>
<td>&quot;valid OOABL path&quot;</td>
<td>Specifies the realm service interface’s class path. SPA security implementation for the REST Web application must specify the <em>IHybridRealm</em> interface class.</td>
</tr>
<tr>
<td>grantedAuthorities</td>
<td>&quot;ROLE_PSCUser&quot;</td>
<td>&quot;ROLE_XXXX&quot;</td>
<td>Specifies the static Spring roles granted for the user if the AppServer realm service interface does not support roles.</td>
</tr>
<tr>
<td>rolePrefix</td>
<td>&quot;ROLE_&quot;</td>
<td>&quot;valid string&quot;</td>
<td>Specifies the role name’s prefix that the Spring security framework must use when converting external role names into Spring roles names.</td>
</tr>
<tr>
<td>roleAttrName</td>
<td>&quot;ATTR.Roles&quot;</td>
<td>&quot;valid string&quot;</td>
<td>Specifies the string name supported by the realm service interface to return a comma-separated list of roles names.</td>
</tr>
<tr>
<td>enabledAttrName</td>
<td>&quot;ATTR_ENABLED&quot;</td>
<td>&quot;valid string&quot;</td>
<td>Specifies the string name supported by the realm service interface to return an account’s enabled state.</td>
</tr>
<tr>
<td>lockedAttrName</td>
<td>&quot;ATTR_LOCKED&quot;</td>
<td>&quot;valid string&quot;</td>
<td>Specifies the string name supported by the realm service interface to return an account’s locked state.</td>
</tr>
<tr>
<td>expiredAttrName</td>
<td>&quot;ATTR_EXPIRED&quot;</td>
<td>&quot;valid string&quot;</td>
<td></td>
</tr>
<tr>
<td>Spring property</td>
<td>Default value</td>
<td>Range of accepted values</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------</td>
<td>--------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>realmPwdAlg</td>
<td>&quot;0&quot;</td>
<td>&quot;0&quot;</td>
<td>&quot;3&quot;</td>
</tr>
<tr>
<td>realmTokenFile</td>
<td>&quot;&quot;</td>
<td>&quot;valid token file&quot;</td>
<td>If a valid token file is provided, it specifies a file name that holds a serialized ClientPrincipal used to authenticate the realm service interface. <strong>Note:</strong> Configure this parameter if you have implemented an extra layer of security, for OERealm, using C-P file and link. The C-P token is generated using genspacp utility.</td>
</tr>
</tbody>
</table>

**Note:** The values that you set for the roleAttrName, enabledAttrName, lockedAttrName, and expiredAttrName properties must match the values of the respective attributes that are used by the OERealm service interface class.

---

**Test the OERealm service**

Once you have configured the SPA security configuration model, you can test the OERealm service.

To test the OERealm service:

1. In the WEB-INF/classes/log4j.properties file, configure the debugging mode of the com.progress.rest.security package to DEBUG or TRACE.
2. Start the Apache Tomcat Java container.
3. Check the contents of the localhost.xxxxx.log file and your OE web application's log file (WEB-INF/adapter/log/xxxx.log) for startup errors related to your SPA security configuration.
4. In a Web browser or any HTTP client, test the user authentication process.
Using SAML in security configurations

Security Assertion Markup Language (SAML) is an XML-based standard for exchanging authentication and authorization data between business entities. It is a product of the OASIS Security Services Technical Committee. SAML specification defines three roles in its security configurations:

- **Principal**: Requests a Web service. Typically, a user.
- **Identity provider** (IdP): Identifies a user and makes the access control decision, that is, only if the user is authenticated with proper permissions to access the Web service.
- **Service provider** (SP): A program or a Web application that can service a Principal's requests. Typically, a Web application.

The following is the representation of a SAML security implementation:

A general scenario implementing SAML for security configuration is as follows:
1. A user (Principal) requests a service from a REST Web application (Service Provider).
2. The REST Web application requests and obtains an identity assertion from the Progress Identity provider (IdP).
3. On the basis of the assertion, if the user has the required permissions for accessing the REST Web application service, the service is performed, otherwise, the service is not performed. Also, in SAML security implementation, one Identity Provider can provide SAML assertions to many Service Providers, and a Service Provider can trust assertions from many independent Identity Providers.

For more information on SAML, see the online resources.

Choosing a SAML security configuration model

Choose and apply one of the SAML security configurations from the available security configurations. For information on security configuration models, see Security configuration models on page 308.
Creating and configuring the Service Provider metadata

For each REST Web application (Service Provider), a metadata file must be generated and sent to all the Identity Providers that the user (Principal) wants to register with. A Service Provider metadata contains keys, services, and URLs that define its SAML endpoints.

**Note:** You can generate a Service Provider metadata file by using the SAML Spring security configuration filter `<security:custom-filter before="FIRST" ref="metadataGeneratorFilter"/>`. This is not supported by an OpenEdge REST Web application. Therefore, you must manually create a Service Provider metadata file.

To manually create and configure a Service Provider metadata file:

1. Extract the `$DLC/servlets/rest/saml.jar` file from the `$DLC/servlets/rest/` directory, and then copy the contents to the `WEB-INF` folder of your REST Web application.
2. Create a new service provider metadata file for your Web application.

**Note:** You must generate your Service Provider metadata as expected by your Identity Provider.

The following is a sample Service Provider metadata file:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<md:EntityDescriptor
xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
entityID="&lt;Entity ID&gt;">
<md:SPSSODescriptor
AuthnRequestsSigned="true" WantAssertionsSigned="true"
protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
<md:KeyDescriptor use="signing">
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:X509Data>
<ds:X509Certificate>
&lt;Certificate information&gt;</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
</md:KeyDescriptor>
<md:NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</md:NameIDFormat>
<md:AssertionConsumerService
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
Location="&lt;Location ID&gt;" index="0" isDefault="true"/>
</md:SPSSODescriptor>
</md:EntityDescriptor>
```
The Service Provider metadata file contains the following definitions (highlighted in bold in the previous codeblock):

<table>
<thead>
<tr>
<th>metadata</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Specify the unique identity of the REST Web application (Service Provider).</td>
</tr>
<tr>
<td>AuthnRequestsSigned</td>
<td>Specify if the REST Web application (Service Provider) signs authentication</td>
</tr>
<tr>
<td></td>
<td>requests.</td>
</tr>
<tr>
<td>WantAssertionsSigned</td>
<td>Specify if the REST Web application requires signed assertions.</td>
</tr>
<tr>
<td>Certificate</td>
<td>Specify the certificate that must be used by the IdP to register the Service</td>
</tr>
<tr>
<td></td>
<td>Provider. This can either be a self-signed or a Certificate Authority (CA)</td>
</tr>
<tr>
<td></td>
<td>signed certificate.</td>
</tr>
<tr>
<td>Binding</td>
<td>Specify the bindings to be included in the metadata for the WebSSO profile.</td>
</tr>
<tr>
<td></td>
<td>Supported values are POST, Artifact, and PAOS. The order of bindings in</td>
</tr>
<tr>
<td></td>
<td>the property determines the order of endpoints in the generated metadata.</td>
</tr>
<tr>
<td></td>
<td>If the IdP does not require keys signed by a specific certification authority, then you can generate your own self-signed key using the Java utility “keytool”.</td>
</tr>
</tbody>
</table>

**Note:** The binding location does not have to be SSL-enabled.

3. Save and close the Service Provider metadata file.

**Note:** This metadata file is used to register the service provider with the IdP.

### Registering the Service Provider with the Identity Provider

After generating the REST Web application (Service provider) metadata file (Creating and configuring the Service Provider metadata on page 337), you must register it with your identity provider (IdP). Each identity provider makes its metadata available for you to import into your service provider application.

An IdP administrator must provide you an IdP metadata file for your service provider based on the configurations that you specified in your service provider metadata file (Creating and configuring the Service Provider metadata on page 337). Therefore, no changes are required in the IdP metadata file that you receive from your IdP administrator.

For instance, if you use the Progress Identity Provider, then you must register your REST Web application metadata file with the Progress Identity Provider and import the Progress identity provider file into your REST Web application as part of SAML specification. The Identity Provider metadata file is in the application WEB-INF folder.

To register your service provider metadata file with your Identity provider:

1. Send the Service Provider metadata file to your IdP administrator.

**Note:** The Service Provider metadata file is required to create an IdP metadata file.

2. Acquire the Identity Provider metadata file from your IdP administrator.

3. Copy the IdP metadata file in the WEB-INF folder of your REST Web application (Service Provider) in your Web server directory.
Configuring the SAML security configuration model

After registering the REST Web application (Service provider) metadata file with the Identity provider, you must configure your SAML security configuration to specify key management, Service provider metadata, and Identity provider metadata details.

To configure your SAML security configuration model:

1. Open the SAML application security configuration model, appSecurity-XXXX-saml.xml, from the WEB-INF folder of your REST Web application in your Web server directory.

2. Search for the following code snippet to configure the key management bean:

   ```xml
   <b:bean id="keyManager" class="org.springframework.security.saml.key.JKSKeyManager">
     <b:constructor-arg value="WEB-INF/samlKeystore.keystore"/>
     <b:constructor-arg type="java.lang.String" value="password"/>
     <b:constructor-arg>
       <b:map>
         <b:entry key="<key-alias>" value="<password>"/>
       </b:map>
     </b:constructor-arg>
     <b:constructor-arg type="java.lang.String" value="<key-alias>"/>
   </b:bean>
   
   Update the following bean properties (highlighted in bold in the previous code block) as follows:

<table>
<thead>
<tr>
<th>Bean property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEB-INF/samlKeystore.keystore</td>
<td>Specify the keystore file that stores the keys for user authentication. This can be a self-signed or a Certificate Authority (CA) generated keystore.</td>
</tr>
<tr>
<td>key-alias</td>
<td>Specify the key alias, that is, a unique keystore name that you set during keystore generation.</td>
</tr>
<tr>
<td>password</td>
<td>Specify the keystore password required to authenticate the REST Web application.</td>
</tr>
</tbody>
</table>

3. Search for the following code snippet to configure the metadata bean:

   ```xml
   <b:bean id="metadata" class="org.springframework.security.saml.metadata.CachingMetadataManager">
     <b:constructor-arg>
       <b:list>
         <!-- IDP Metadata file -->
         <b:bean class="org.springframework.security.saml.metadata.ExtendedMetadataDelegate">
           <b:constructor-arg>
             <b:bean class="org.opensaml.saml2.metadata.provider.FilesystemMetadataProvider">
               <b:constructor-arg>
                 <b:value type="java.io.File">IdP metadata file name</b:value>
               </b:constructor-arg>
               <b:property name="parserPool" ref="parserPool"/>
             </b:bean>
           </b:constructor-arg>
         </b:bean>
       </b:list>
     </b:constructor-arg>
   </b:bean>
   ```
Update the following bean properties (highlighted in bold in the previous code block) as follows:

<table>
<thead>
<tr>
<th>Bean property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDP metadata file name</td>
<td>Specify the location of the IdP metadata file.</td>
</tr>
<tr>
<td>Service provider metadata file name</td>
<td>Specify the location of the REST Web application (Service provider) metadata file.</td>
</tr>
</tbody>
</table>

4. Search for the following code snippet to configure the processing filter bean:

```xml
<! -- Processing filter for WebSSO profile messages -->
<bean id="samlProcessingFilter"
     class="com.progress.rest.security.OESamlProcessingFilter">
    <property name="samlEntryPointHandler" ref="initializeSAMLEntryPoint"/>
    <property name="authenticationManager" ref="RestApplicationAuth"/>
    <property name="authenticationSuccessHandler" ref="successRedirectHandler"/>
</bean>
```
Update the following bean property (highlighted in bold in the previous code block) as follows:

<table>
<thead>
<tr>
<th>Bean property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultFilterProcessesUrl</td>
<td>Specify, as a value, the URL where the processing filter for WebSSO must be applied.</td>
</tr>
<tr>
<td>defaultIdpProcessingUrl</td>
<td>Specify, as a value, your IdP’s URL. For instance, if you use the Progress IdP, you specify the value as <a href="https://secure-test.progress.com">https://secure-test.progress.com</a>.</td>
</tr>
</tbody>
</table>

5. Start your Tomcat Web server, in which this REST Web application is deployed, for the security configurations to take effect.
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 an OpenEdge Web service whether you deploy it as a Web service using an instance of the Web Services Adapter (WSA) or REST Management Agent 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.

For details, see the following topics:

- **Overview**
- **Alphabetical reference**

**Overview**

You can set default values for Web service properties on a WSA instance or a REST Web application instance using OpenEdge Management or OpenEdge Explorer, which then apply to every Web service that you deploy to that WSA instance or the REST Management Agent 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 the WSA Adapter, REST Management Agent, or the OpenEdge Adapter for Sonic ESB.

**Summary of properties**

The following table lists a summary of all the properties that you can set for a service, depending on its session model (free, managed, or both).
Table 93: Service properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Session model</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appServerKeepAlive on page 348</td>
<td>Both</td>
<td>Indicates if the client would like to employ the AppServer Keepalive protocol on this connection.</td>
</tr>
<tr>
<td>appServiceConnectionMode on page 348</td>
<td>Both</td>
<td>Indicates whether this connection is session-free or session-managed.</td>
</tr>
<tr>
<td>appServiceHost on page 348</td>
<td>Both</td>
<td>Host name for the NameServer or AppServer that supports an application service.</td>
</tr>
<tr>
<td>appServiceName on page 348</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 on page 348</td>
<td>Both</td>
<td>The port number to access the application service bound by the service.</td>
</tr>
<tr>
<td>appServiceProtocol on page 348</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>clientASKActivityTimeout on page 349</td>
<td>Both</td>
<td>Specifies the duration for which the AppServer takes active steps to determine if the client is still connected. The ClientASK protocol denotes that the direction of the keepalive messages is from the client to the servers.</td>
</tr>
<tr>
<td>clientASKResponseTimeout on page 349</td>
<td>Both</td>
<td>Specifies the duration for which the AppServer can receive messages from the server. The ClientASK protocol denotes that the direction of the keepalive messages is from the client to the servers.</td>
</tr>
<tr>
<td>connectionLifetime on page 349</td>
<td>Free</td>
<td>The maximum lifetime (in seconds) of AppServer connections in the connection pool for this service.</td>
</tr>
<tr>
<td>connectionTimeout on page 349</td>
<td>Both</td>
<td>Specifies the duration, in Integer, for which a client must stay connected to the session-managed or session-free AppServer.</td>
</tr>
<tr>
<td>idleSessionTimeout on page 350</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 on page 350</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>Property</td>
<td>Session model</td>
<td>Short description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>maxSessions</strong> on page 350</td>
<td></td>
<td>The maximum number of connected sessions allowed in the service connection pool</td>
</tr>
<tr>
<td><strong>minIdleConnections</strong> on page 350</td>
<td>Free</td>
<td>The minimum number of idle, or inactive, AppServer connections to maintain</td>
</tr>
<tr>
<td><strong>minSessions</strong> on page 351</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><strong>noHostVerify</strong> on page 351</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><strong>noSessionReuse</strong> on page 351</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><strong>nsClientMaxPort</strong> on page 351</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><strong>nsClientMinPort</strong> on page 351</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><strong>nsClientPicklistExpiration</strong> on page 352</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><strong>nsClientPicklistSize</strong> on page 352</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><strong>nsClientPortRetry</strong> on page 352</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><strong>nsClientPortRetryInterval</strong> on page 352</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><strong>requestWaitTimeout</strong> on page 352</td>
<td>Free</td>
<td>Determines how the adapter handles requests when the service connection pool becomes full</td>
</tr>
</tbody>
</table>
**Short description**

**Session model**

**Short description**

<table>
<thead>
<tr>
<th>Property</th>
<th>Session model</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serviceAvailable on page 353</td>
<td>Both</td>
<td>Specifies whether the client can access the enabled Web service applications.</td>
</tr>
<tr>
<td>serviceFaultLevel on page 353</td>
<td>Both</td>
<td>The degree of detail returned to the client for a service fault.</td>
</tr>
<tr>
<td>serviceLogEntryTypes on page 353</td>
<td>Both</td>
<td>Specifies the type of information recorded when performing diagnostic logging. The value is a colon-separated list of valid entry types.</td>
</tr>
<tr>
<td>serviceLoggingLevel on page 354</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>
<tr>
<td>sessionMode on page 354</td>
<td>Both</td>
<td>Indicates whether this connection is session-free or session-managed.</td>
</tr>
<tr>
<td>stale04GLObjectTimeout on page 354</td>
<td>Both</td>
<td>The maximum duration (in seconds) that a service object can be idle before it is released</td>
</tr>
<tr>
<td>waitIfBusy on page 355</td>
<td>Managed</td>
<td>Determines how to handle client requests to a service that is busy processing a prior request</td>
</tr>
</tbody>
</table>

**Note:** 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 Deploying and Managing OpenEdge Web Services on page 107.

**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 on page 353
- serviceLoggingLevel on page 354

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.
appServerKeepAlive

**Session model:** Managed or Free

Indicates that the client would like to employ the AppServer keepalive protocol on a connection, if supported and enabled by the AppServer or the client. To enable the protocol, specify the `allowServerASK` or `allowClientASK` value. To disable the protocol, specify `denyServerASK` or `denyClientASK` value.

The absence of this property indicates that the default value will be used on the connection. You must set this property only if your AppServer is operating in Stateless or State-free operating mode.

**Installation default:** denyClientASK, allowServerASK

appServiceConnectionMode

**Session model:** Managed or Free

Indicates whether the connection is session-managed or session-free. Set the property to 1 for session-free connection type or set it to 0 for session-managed connection type.

**Installation default:** 1

appServiceHost

**Session model:** Managed or Free

Host name for the NameServer or AppServer that supports an application service with the name specified by `appServiceName` on page 348 property.

**Installation default:** localhost

appServiceName

**Session model:** Managed or Free

The name of an application service supported by the NameServer or AppServer specified by `appServiceHost` on page 348 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 `appServiceHost` on page 348 property.

**Installation default:** 5162

appServiceProtocol

**Session model:** Managed or Free
The protocol used to access the host specified by appServiceHost on page 348 property. For a NameServer host, this is "AppServer"; for an AppServer host, "AppServerDC".

Installation default: AppServer

**clientASKActivityTimeout**

Session model: Managed or Free

Specifies the time, in seconds, for which a client must wait without activity from the server before sending an askPing message to the AppServer. This message is sent to the AppServer to determine if the server is still alive. The clientASK protocol denotes that the direction of the askPing keepalive message is from the client to the server.

This property is active only if your AppServer is operating in the stateless or state-free operating mode.

Installation default: 60

**clientASKResponseTimeout**

Session model: Managed or Free

Specifies the time, in seconds, for which a client must wait for a response for an askPing message that has been sent to the AppServer. If a response is not received in the set time, the client concludes that the connection has failed and performs an action to overcome a fatal communication error. The clientASK protocol denotes that the direction of the askPing keepalive message is from the client to the server.

This property is active only if your AppServer is operating in the Stateless or State-free operating mode.

Installation default: 60

**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 on page 350. A positive value indicates that the WSA Adapter, REST Management Agent, or the OpenEdge Adapter for Sonic ESB maintains any AppServer connections for this service for the specified number of seconds. Thus, when idleSessionTimeout on page 350 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

**connectionTimeout**

Session model: Managed and Free

Specifies the duration, in milliseconds, for which a client waits for a successful connection with the session-managed or session-free AppServer. If the connection is not successful within the defined limit, the client throws a connection-timeout exception. You can use this property when you want a client program to disconnect from a long connection request.
Installation default: 0 (indefinite wait)

**idleSessionTimeout**

*Session model:* Free only

The duration (in seconds) between attempts by the WSA Adapter, REST Management Agent, or the 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 on page 349 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, REST Management Agent, or the OpenEdge Adapter for Sonic ESB. This value must be between the value of minSessions on page 351 property and the maxSessions on page 350 property, inclusive, unless maxSessions on page 350 is set to 0. If maxSessions on page 350 is set to 0, the initialSessions value must only be greater than or equal to minSessions on page 351.

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 on page 350, the WSA Adapter, REST Management Agent, or the OpenEdge Adapter for Sonic ESB creates no additional sessions for this service, and handles all requests for this service according to requestWaitTimeout on page 352 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, REST Management Agent, or the 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, REST Management Agent, or the 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 `appServiceProtocol` on page 348 property). If cleared, the Web service 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 `appServiceProtocol` on page 348 property).

**Installation default:** 0

nsClientMaxPort

**Session model:** Managed or Free

The maximum value for the UDP port number that the WSA Adapter, REST Management Agent, or the OpenEdge Adapter for Sonic ESB specifies when communicating with the NameServer. This value must be greater than or equal to the value of `nsClientMinPort` on page 351 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, REST Management Agent, or the 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 `nsClientMaxPort` on page 351 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, REST Management Agent, or the 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, REST Management Agent, or the 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, REST Management Agent, or the 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, REST Management Agent, or the 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, REST Management Agent, or the 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 on page 350, and all sessions are currently running requests.)

The adapter handles such requests according to the value of requestWaitTimeout, as shown in the following table.

Table 94: requestWaitTimeout property values

<table>
<thead>
<tr>
<th>If the value is . . .</th>
<th>The WSA Adapter, REST Management Agent, or the 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

**serviceAvailable**

**Session model:** Managed or Free

Specifies if the client is allowed to or not allowed to access the Web service. Set to 1 to allow access. Set to 0 to prevent access.

**Note:** When you work with a REST Web application, you must set the serviceAvailable property using the commands RESTMAN enable and RESTMAN disable. For more information on the commands, see Using the RESTMAN Utility on page 303

Installation default: 0

**serviceFaultLevel**

**Session model:** Managed or Free

The degree of detail returned to the client for a Web service fault as determined by an integer value. A level of 2 returns basic information in the <FaultCode> and <FaultString> elements for each Web service 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

**serviceLogEntryTypes**

**Session model:** Managed or Free
Specifies the type of information recorded during diagnostic logging. The value is a colon-separated list of valid entry types.

For information on Web service logging in OpenEdge, see *OpenEdge Development: Debugging and Troubleshooting*.

**Note:** For REST Web applications, the installation default is **REST**. You can change the log entry type to BrokerClient, PoolMgmt, RunProcs, SessionPool, and RefCounts.

### serviceLoggingLevel

**Session model:** Managed or Free

The amount and type of information written for each service log entry by the WSA Adapter, REST Management Agent, or the OpenEdge Adapter for Sonic ESB to the adapter log file, cumulatively determined by the integer values from 1 to 4, as shown in the following table.

For information on logging in OpenEdge, see *OpenEdge Development: Debugging and Troubleshooting*.

#### Table 95: serviceLoggingLevel property values

<table>
<thead>
<tr>
<th>If the value is . . .</th>
<th>The WSA Adapter, REST Management Agent, or the 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 the Web service faults not included)</td>
</tr>
<tr>
<td>2</td>
<td>Additional errors that result in the Web service 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

### sessionMode

**Session model:** Managed and Free

Indicates whether the connection is session-managed or session-free. Set the property to 1 for session-free connection type or set it to 0 for session-managed connection type.

Installation default: 0

### 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, REST Management Agent, or the 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, REST Management Agent, or the 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.

For details, see the following topics:

- ADAPTCOMFIG
- ADAPTMAN
- AIACONFIG
- ASBMAN
- ASCONFIG
- ESBOEGEN
- NSCONFIG
- NSMAN
- PROADSV
- RESTGEN
- RESTMAN deploy
- RESTMAN disable
- RESTMAN enable
• RESTMAN getdefaults
• RESTMAN getprops (Application)
• RESTMAN getprops (Management Agent)
• RESTMAN getstats (Application)
• RESTMAN getstats (Management Agent)
• RESTMAN list
• RESTMAN query (Application)
• RESTMAN query (Management Agent)
• RESTMAN resetdefaults (Management Agent)
• RESTMAN resetprops
• RESTMAN resetstats (Application)
• RESTMAN republish (Application)
• Deleting a REST Web application's files
• RESTMAN resetstats (Management Agent)
• RESTMAN setdefault (Management Agent)
• RESTMAN setprops (Application)
• RESTMAN setprops (Management Agent)
• RESTMAN undeploy
• RESTMAN unpublish (Application)
• 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 [ [ -name adapter-broker ] [ -propfile path-to-properties-file ] [ -validate ] ]</td>
</tr>
<tr>
<td>Windows</td>
<td>adaptconfig [ [ -name adapter-broker ] [ -propfile path-to-properties-file ] [ -validate ] ]</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.
Notes
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>UNIX</td>
<td>adaptman {</td>
</tr>
<tr>
<td>Windows</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>
</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.

Notes

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.

Example

The following table shows several examples that use the adaptman command to start an instance called SonicMQ1.

Table 96: ADAPTMAN command examples

<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 whose AdminServer is on port 12935</td>
<td>adaptman -host xxxxxx -port 12935 -i sonicMQ1 -q</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</td>
<td>`aiaconfig [ [ [ -name AIA-instance-name] [ -propfile path-to-properties-file] [ -validate ] ]</td>
</tr>
<tr>
<td>Windows</td>
<td></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.

### Notes

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

Starts, stops, adds AppServer agents, trims AppServer agents, refreshes an AppServer and queries the status for an AppServer instance, its AppServer agent and also queries the status of AppServer Refresh command.
**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| UNIX             | asbman {
| Windows          |   -name AppServer-name
|                  |   { -kill | -start | -stop | -query |
|                  |       -addservers number-to-start |
|                  |       -trimservers number-to-trim |
|                  |       -listclients |
|                  |       -clientdetail connection-handle |
|                  |       -listallprops |
|                  |       -refresh |
|                  |       -refreshStatus |
|                  |       -cancelRefresh }
|                  | [ -host host-name -user user-name | -user user-name]
|                  |   [ -port port-number]
|                  | } | -help } |

- **-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. The following table describes the information provided by the command.

**Table 97: 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 the following table.
<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. This is usually the same value that is accessible to the ABL client application using the \texttt{CLIENT-CONNECTION-ID} attribute on the server object handle, and to the ABL server application using the \texttt{SERVER-CONNECTION-ID} attribute on the session handle.</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
```

-\texttt{-listallprops}

Displays all active broker properties, including updated values for dynamic properties that have changed.

-\texttt{-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.

Example

The following table shows several examples that use the ASBMAN command. Assume the AppServer instance is AS1 and the NameServer is NS1.
### Table 99: 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. | `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`                                              |

- **-agentdetail**

  Displays the procedure run by an AppServer agent. You can select the specific AppServer agent by specifying the process ID (PID) of an agent process.

- **Example**

  Use the following command to display the .r procedure run by an AppServer agent. It displays the port and connection state of an AppServer agent.

  ```
  asbman -name <name> -agentdetail <pid>
  ```

- **-agentkill**

  Stops a specific AppServer agent forcefully. The PID specified to the `asbman` command must be an agent process for the specified AppServer. Use the `kill` command to stop a specific AppServer agent on UNIX. Use the `taskkill` command to stop a specific AppServer agent on Windows Professional Edition. This utility is not supported by the Windows Home Edition.

- **Example**

  Use the following command to stop an AppServer agent forcefully.

  ```
  asbman -name <name> -agentkill <pid>
  ```

- **-agentstop**

  Stops an AppServer agent gracefully.

---

18 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 Management or 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.
Example

Use the following command to stop an AppServer agent.

```
asbman -name <name> -agentstop <pid>
```

`-refresh`

Allows the users to trim individual agents and restart new ones in order to gradually migrate to a new version of r-code from their current version. Details of a Refresh operation are logged at a verbose level.

Example

Use the following command to refresh an AppServer.

```
asbman -name <name> -refresh
```

`-cancelRefresh`

Cancels a refresh command. This command cancels a refresh operation that is running for too long. It will also stop any busy agent marked for refresh from getting trimmed automatically. The broker will not add new agents for any agent that were trimmed when the cancelled refresh operation was running. An administrator can add more agents if needed. After the refresh is cancelled, a message is returned indicating that the refresh was successful. If no refresh was active for that broker, the following message is returned:

Refresh for <brokerName> is not active

Example

Use the following command to cancel the refresh command issued for an AppServer.

```
asbman -name <name> -cancelRefresh
```

`-refreshStatus`

This command returns detailed information that is captured in the following table:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refresh Active</td>
<td>Yes/No</td>
<td>If the refresh is not active, and if there was a refresh operation issued earlier, it shows the refreshStatus of that operation</td>
</tr>
<tr>
<td>Field</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Refresh Start Time</td>
<td>yyyy-MM-dd'T'HH:mm:ss.SSS</td>
<td>Start time of the currently executing refresh operation. If no refresh is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>active, it shows the start time of the latest refresh operation that was</td>
</tr>
<tr>
<td></td>
<td></td>
<td>completed</td>
</tr>
<tr>
<td>Refresh End Time</td>
<td>yyyy-MM-dd'T'HH:mm:ss.SSS</td>
<td>End time of the latest refresh operation that was completed. This is not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>applicable for a running refresh operation</td>
</tr>
<tr>
<td>Refresh Cancel Time</td>
<td>yyyy-MM-dd'T'HH:mm:ss.SSS</td>
<td>The time when a refresh operation was cancelled</td>
</tr>
<tr>
<td>Agents at refresh start</td>
<td>Integer value &gt;= 0</td>
<td>Number of agents that were running at the time when refresh command was</td>
</tr>
<tr>
<td></td>
<td></td>
<td>issued</td>
</tr>
<tr>
<td>Agents marked for refresh</td>
<td>Integer value &gt;= 0</td>
<td>Number of agents that are marked to be refreshed. Agents that are busy will</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be marked for refresh and will be stopped only after they become available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Also the agents that were added after refresh operation started will not be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>refreshed. This field will be displayed for active refresh operation only.</td>
</tr>
<tr>
<td>New agents added during refresh</td>
<td>Integer value &gt;= 0</td>
<td>No. of agents that got added after the refresh was issued. These agents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>won't be running old r-code, so they will not be marked for refresh</td>
</tr>
<tr>
<td>Agents at refresh cancel</td>
<td>Integer value &gt;= 0</td>
<td>No. of agents running at the time of refresh cancellation</td>
</tr>
<tr>
<td>Agents refreshed</td>
<td>Integer value &gt;= 0</td>
<td>It shows the actual number of agents refreshed for a refresh operation that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is complete</td>
</tr>
<tr>
<td>Refresh operation count</td>
<td>Integer value &gt;= 0</td>
<td>Number of times a refresh operation is invoked including the currently</td>
</tr>
<tr>
<td></td>
<td></td>
<td>executing refresh operation since start of the broker</td>
</tr>
<tr>
<td>Refresh cancel count</td>
<td>Integer value &gt;= 0</td>
<td>Number of times a refresh cancellation was is invoked since start of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>broker</td>
</tr>
</tbody>
</table>
**Field** | **Value** | **Description**
---|---|---
PID of agents marked for refresh | Integer value >= 0 | Number of agents added after the refresh was issued. This doesn't include the agents that were started as replacement agents. This include agents added due to addservers or new client requests.

PID of agents refreshed | Integer value >= 0 | List the PID of agents that were Refreshed(replaced by new agents).

**Example**

Use the following command to query the status of a refresh command issued for an AppServer.

```
asbman -name <name> -refreshStatus
```

**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>
<tbody>
<tr>
<td>UNIX</td>
<td>`asconfig [ [ -name AppServer-name ] [ -propfile path-to-properties-file ] [ -validate ] ]</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
</tbody>
</table>

- **-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.

Notes

Never needed if you always use OpenEdge Management or 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>
<tbody>
<tr>
<td>Windows</td>
<td>esboegen [-source directory]</td>
</tr>
<tr>
<td></td>
<td>[-esboe directory] [-archive filename]</td>
</tr>
<tr>
<td></td>
<td>[-rcode directory]</td>
</tr>
<tr>
<td></td>
<td>[-recurse]</td>
</tr>
<tr>
<td></td>
<td>[files]</td>
</tr>
</tbody>
</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

• Progress Developer Studio for OpenEdge 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

NSCONFIG

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>
</table>
| Windows          | nsconfig [[ -name name-server] [-propfile path-to-properties-file] [-validate] | -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 Management or OpenEdge Explorer.
- A single NameServer can simultaneously support all of the AppServer, WebSpeed, and DataServer products using OpenEdge Management or OpenEdge Explorer.

Example

The following table shows two examples that use the NSCONFIG command. Assume the NameServer is NS1.
Table 101: 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.

NSMAN

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>
</table>
| Windows          | nsman {  
|                  |   { -name nameserver  
|                  |     { -kill | -start | -stop | -query | -listallprops }  
|                  |       [ -host host-name -user user-name | -user user-name]  
|                  |         [ -port port-number]  
|                  | } | -help } |

-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

Specify 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.

Example

The following table 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 102: 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 NameServer(^{19})</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 NameServer(^{19})</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 NameServer(^{19})</td>
<td>nsman -name NS1 -host finance -port 9999 -user tom -stop</td>
</tr>
</tbody>
</table>

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>
</tbody>
</table>

- **-start**
  
  Starts the AdminServer.

\(^{19}\) Prompts for a password.
-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 7844.

-help

Displays the command-line help.

Example

The following table shows several examples that use the NSCONFIG command. Assume the NameServer is NS1.

Table 103: PROADSV command examples

<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.

RESTGEN

Generates a Web application (WAR) or Web services (PAAR) file for a REST Web application.
## Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| UNIX             | restgen
|                  | -proj location-of-the-REST/Mobile-project
|                  | { [restSvcNames REST-service-name-1:
|                  |     REST-service-name-2:
|                  |     REST-service-name-3...]
|                  | [MobSvcNames Mobile-service-name-1:
|                  |     Mobile-service-name-2:
|                  |     Mobile-service-name-3...]
|                  | [MobApps Mobile-application-name-1:
|                  |     Mobile-application-name-2:
|                  |     Mobile-application-name-3...]
|                  | }
|                  | -target location-for-the-generated-war/zip-file
|                  | [-includeJars {true | false}]
|                  | {genPaar | genMobPaar | genRESTWar | genMobWar | genMobAppWar | genOnlyMobApp} |
| Windows          |        |

- **-proj location-of-the-REST-project**

  The location of the REST or Mobile project folder.

- **-restSvcNames REST-service-name-1:REST-service-name-2...**

  Colon-delimited names of the REST services to be generated in a REST project.

- **-MobSvcNames Mobile-service-name-1:Mobile-service-name-2...**

  Colon-delimited names of the Mobile services to be generated in a REST project.

- **-MobApps Mobile-application-name-1:Mobile-application-name-2...**

  Colon-delimited names of the Mobile Web application to be generated in a REST project.
-target location-for-the-generated-war/zip-file

The location where the RESTGEN generates the REST Web application (WAR) file or a REST/Mobile Web Service (PAAR). In case of REST/Mobile Web Service, the PAAR file is generated as a ZIP file. If unspecified, the RESTGEN utility generates the output in %WRKDIR%.

-excludeJars { true | false }

Specifies whether all the REST/Mobile Application library files must be regenerated or not. If unspecified, the library files are regenerated, that is, -includeJars is set to true.

{ genPaar | genMobPaar | genRESTWar | genMobWar | genMobAppWar | genOnlyMobApp }

The following section describes how and when to use the above parameters:

- genPaar: Generates a ZIP file that contains REST services—specified in the -restSvcNames parameter—for a REST project.

  The ZIP file maintains the REST project folder structure. Typically, the ZIP file's folder structure for a REST service is WEB-INF/adapter/rest-service/rest-service.paar.

- genMobPaar: Generates a ZIP file that contains Mobile services, specified in the -MobSvcNames parameter, for a Mobile project.

  The ZIP file maintains the Mobile project folder structure. Typically, the ZIP file contains WEB-INF/adapter/mobile-service/mobile-service.paar and the catalog files for each mobile service, static/catalog-name.json.

- genRESTWar: Generates a deployable Web application for a REST project.

  The Web application consists of the REST services that are specified in the -restSvcNames parameter.

- genMobWar: Generates a deployable Mobile Web application for a mobile project.

  The Mobile Web application consists of the Mobile services that are specified in the -mobSvcNames parameter.

  Note: The generated mobile application does not contain the Mobile UI App.

- genOnlyMobAppWar: Generates a Mobile Web application for a mobile project.

  The Mobile Web application consists of the Mobile Apps that are specified in the -mobApps parameter.

  Note: The generated mobile application contains only the Mobile UI Apps.

- genMobAppWar: Generates a Mobile Web application for a mobile project.

  The Mobile Web application consists of both the Mobile services that are specified in the -mobSvcNames parameter, and the Mobile apps that are specific in the -mobApps parameter.
Notes

- You can use the RESTGEN utility to generate a ZIP file with REST or Mobile services, and then use the ZIP file as an input file in the RESTMAN republish utility. For more information on republishing a REST Web application, see Republishing a REST Web application on page 279.

- The following examples show how the different WAR and PAAR generation options in RESTGEN utility work:

  In the following example, a RESTProject.zip is created in the location C:/OpenEdge. The ZIP file contains the REST services, ServiceA and ServiceB, of a REST Web application, RESTProject:

  ```bash
  restgen -proj C:/RESTProject -restSvcNames ServiceA:ServiceB -target C:/OpenEdge/RESTProject.zip -genPaar
  ```

  In the following example, a MobileProject.zip is created in the location C:/OpenEdge. The ZIP file contains the Mobile services, ServiceA and ServiceB, of a Mobile Web application, MobileProject:

  ```bash
  restgen -proj C:/MobileProject -MobSvcNames ServiceA:ServiceB -target C:/OpenEdge/MobileProject.zip -genMobPaar
  ```

  In the following example, a REST Web application, RESTProject.war, is created in the location C:/OpenEdge. The WAR file contains the REST services, ServiceA and ServiceB:

  ```bash
  restgen -proj C:/RESTProject -restSvcNames ServiceA:ServiceB -target C:/OpenEdge/RESTProject.war -genRESTWar
  ```

  In the following example, a Mobile Web application, MobileProject.war, is created in the location C:/OpenEdge. The WAR file contains the Mobile services, ServiceA and ServiceB:

  ```bash
  restgen -proj C:/MobileProject -MobSvcNames ServiceA:ServiceB -target C:/OpenEdge/MobileProject.war -genMobWar
  ```

  In the following example, a Mobile Web application, MobileProject.war, is created in the location C:/OpenEdge. The WAR file contains the Mobile apps, MobAppA and MobAppB:

  ```bash
  restgen -proj C:/MobileProject -MobApps MobAppA:MobAppB -target C:/OpenEdge/MobileProject.war -genOnlyMobApp
  ```

  In the following example, a Mobile Web application, MobileProject.war, is created in the location C:/OpenEdge. The WAR file contains the Mobile apps, MobAppA and MobAppB, and the Mobile services, ServiceA and ServiceB:

  ```bash
  restgen -proj C:/MobileProject -MobSvcNames ServiceA:ServiceB -MobApps MobAppA:MobAppB -target C:/OpenEdge/MobileProject.war -genMobAppWar
  ```
RESTMAN deploy

Deploys a REST Web application (WAR) on the Web server.

This command transfers the REST Web application's WAR file to the servlet container on which the REST Management Agent is installed.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman -name instance-name -war war-location-on-restman-machine [ -appname application-name ] -deploy</td>
</tr>
<tr>
<td>Windows</td>
<td>restman -name instance-name -war war-location-on-restman-machine [ -appname application-name ] -deploy</td>
</tr>
</tbody>
</table>

- **-name instance-name**
  
  The name of the REST Management Agent instance.

- **-war war-location-on-restman-machine**
  
  The path of the WAR file on the RESTMAN machine.

- **-appname application-name**
  
  The name of the REST Web application that you want to deploy on the Web server. This replaces the value in the WAR file.

Notes

- The WAR file must be accessible from the machine running OpenEdge Management, OpenEdge Explorer, or the RESTMAN utility; that is, located on a local drive or a mapped network drive.
- When a REST Web application is deployed, it is initially disabled.
- All the properties of the Rest Management Agent are in its default.props file which is used to initialize the runtime.props file of each REST Web application deployed on the Web server.

RESTMAN disable

Makes a deployed REST Web application unavailable to incoming client requests by setting the serviceAvailable property of the REST Web application to 0.
Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman</td>
</tr>
<tr>
<td></td>
<td>-name instance-name</td>
</tr>
<tr>
<td></td>
<td>-appname application-name</td>
</tr>
<tr>
<td></td>
<td>-disable</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

-name instance-name

The name of the REST Management Agent instance.

-appname application-name

The name of the REST Web application that you want to disable.

Notes

When a REST Web application is disabled, the REST Management Agent begins a shutdown process that involves stopping the Web service's existing client requests and terminating any AppServer connections in the application's connection pool.

Although a REST Web application is disabled, it can return customer HTTP response codes and text. This can be configured through the REST Management Agent.

RESTMAN enable

Makes a deployed REST Web application available to incoming client requests by setting the serviceAvailable property of the REST Web application to 1.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman</td>
</tr>
<tr>
<td></td>
<td>-name instance-name</td>
</tr>
<tr>
<td></td>
<td>{ -appname application-name }</td>
</tr>
<tr>
<td></td>
<td>-enable</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

-name instance-name

The name of the REST Management Agent instance.

-appname application-name

The name of the REST Web application that you want to enable.

Notes

- When a Web service is deployed, it is disabled.
If you set or reset any application properties, the changes to the properties take effect only when the REST Web application is enabled.

When a Web service is enabled, the REST Management Agent begins a startup process that involves establishing the Application's connection pool, which, in turn, involves pre-establishing any AppServer connections that will be cached to handle incoming client requests more efficiently.

**RESTMAN getdefaults**

**Displays the default properties associated with the REST Management Agent.**

All the properties of the Rest Management Agent in its default.props file are used to initialize the runtime.props file of each REST Web application deployed on the Web server.

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman -name instance-name -getdefaults</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
</tbody>
</table>

-name instance-name

The name of the REST Management Agent instance.

For more information on REST Management Agent properties, see Reference to OpenEdge Web Service Properties on page 343.

**RESTMAN getprops (Application)**

**Displays all the properties of a REST Web application.**

All the properties of an Application are in its runtime.props file.

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman -name instance-name -appname application-name -getprops</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
</tbody>
</table>

-name instance-name

The name of the REST Management Agent instance.
RESTMAN getprops (Management Agent)

Displays the current value of the properties of Rest Management Agent that can be changed while the Agent is running.

The REST Management Agent’s properties reside in the oerm.properties file. By default, the oerm.properties file is located in <web server directory>\oerm\WEB-INF\oerm.data\.

The following are the properties you can get for a REST Management Agent:

- **adminEnabled** — Specifies whether administrator privileges are in effect, either enabled or disabled.
- **description** — Specifies the description of the REST Management Agent instance.
- **logAppend** — Specifies whether to start a new log file each time the server is restarted, either true or false.
- **logEntryTypes** — Specifies a list of logging entry types separated by commas.
- **logFile** — Specifies the location and name of the REST Management Agent log file.
- **loggingLevel** — Specifies the amount of information to be written to the broker log.
- **managerPropertiesURI** — Specifies the identifier for the REST Management Agent resource. This is a read-only property.
- **name** — Specifies the name of the REST Management Agent.
- **propertiesFileName**— Specifies the path and name of the oerm.properties file.
- **webAppEnabled** — Controls the REST Management Agent's ability to accept and process requests to any of its deployed REST Web applications.

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td><code>restman</code></td>
</tr>
<tr>
<td></td>
<td><code>-name instance-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-getprops</code></td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
</tbody>
</table>

- `-name instance-name`

The name of the REST Management Agent instance.

RESTMAN getstats (Application)

Displays the statistics for a REST Web application, such as Number of Successful User Requests and Average Response Time for User Requests.
The following is the list of REST Web application statistics:

- **Adapter errors** — Total number of REST failures
- **Application errors** — Total number of errors related to REST Web application
- **Average response time** — Average response time of each request response.
- **Failed admin logins** — Total number of failed administrator logins
- **Failed admin requests** — Total number of administrator requests that failed due to missing request information
- **Failed app server connections** — Total number of failed AppServer connections in the connection pool
- **Failed user logins** — Total number of failed user logins
- **Failed user requests** — Total number of requests that failed due to missing request information
- **Max response time** — Maximum response time of each request response
- **Successful admin logins** — Total number of successful administrator logins
- **Successful admin requests** — Total number of successful administrator requests
- **Successful app server connections** — Total number of successful AppServer connections in the connection pool for this REST Web application
- **Successful user logins** — Total number of successful user logins
- **Successful user requests** — Total number of successful user requests handled by the selected REST Web application
- **Creation time** — Date and time that counting began for the statistics
- **Start time** — Start date and time of the REST Web application

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td><code>restman -name instance-name</code></td>
</tr>
<tr>
<td>Windows</td>
<td><code>restman -name instance-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-appname application-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-getstats</code></td>
</tr>
</tbody>
</table>

`-name instance-name`

The name of the REST Management Agent instance.

`-appname application-name`

The name of the REST Web application.

For more information on these statistics, see the *OpenEdge Management and OpenEdge Explorer: Configuration* manual or the OpenEdge Management or OpenEdge Explorer online help.
RESTMAN getstats (Management Agent)

Displays the statistics for a REST Management Agent.
The following are the statistics you can get for a REST Management Agent:

- **Active Requests** — Requests currently queued and being acted upon by the REST Management Agent
- **Authentication Errors** — Total number of authentication errors
- **HTTP Request Errors** — Requests received from the HTTP listener, including administrative, and REST Management Agent requests
- **HTTP Requests** — Total requests received from the HTTP listener, including administrative, and REST requests
- **Method not allowed errors** — Total errors returned by the REST Management Agent, with error counts broken out at the bottom of the list for each of several error categories when total errors are greater than zero (0)
- **REST Disabled errors** — calls while REST Management Agent is disabled
- **Services disabled** — REST Web applications deployed to this REST Management Agent that are disabled from client access
- **URL not found errors** — calls made to unidentified resources
- **Creation Time** — Date and time that counting began for the statistics
- **Start Time** — Start Date and time of the REST Management Agent instance

**Syntax**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman</td>
</tr>
<tr>
<td>Windows</td>
<td>-name instance-name</td>
</tr>
<tr>
<td></td>
<td>-getstats</td>
</tr>
</tbody>
</table>

- **-name instance-name**

    The name of the REST Management Agent instance.

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

RESTMAN list

Displays the list of REST Web applications deployed by the REST Management Agent instance on the Web server.

For each REST Web application, the following details are displayed:

- Application name
RESTMAN query (Application)

Displays the following information about a REST Web application:

- Target URI and Statistics URI
- Status
- Relevant application properties

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman -name instance-name</td>
</tr>
<tr>
<td>Windows</td>
<td>restman -name instance-name -list</td>
</tr>
</tbody>
</table>

- name instance-name

The name of the REST Management Agent instance.

RESTMAN query (Application)

Displays the following information about a REST Web application:

- Target URI and Statistics URI
- Status
- Relevant application properties

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman -name instance-name -appname application-name -query</td>
</tr>
<tr>
<td>Windows</td>
<td>restman -name instance-name -appname application-name -query</td>
</tr>
</tbody>
</table>

- name instance-name

The name of the REST Management Agent instance.

- appname application-name

The name of the REST Web application.

RESTMAN query (Management Agent)

Displays the following characteristics and properties of the REST Management Agent:

- Status
- REST Administration and the REST Services status
Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman -name instance-name -query</td>
</tr>
<tr>
<td>Windows</td>
<td>restman -name instance-name -resetdefaults</td>
</tr>
</tbody>
</table>

- name instance-name

The name of the REST Management Agent instance.

RESTMAN resetdefaults (Management Agent)

Resets the properties of a Rest Management Agent to its default values.

All the properties of the Rest Management Agent that are used when initializing a REST Web application are in its default.props file.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman -name instance-name -resetdefaults</td>
</tr>
<tr>
<td>Windows</td>
<td>restman -name instance-name -resetdefaults</td>
</tr>
</tbody>
</table>

- name instance-name

The name of the REST Management Agent instance.

For information on the properties, see Reference to OpenEdge Web Service Properties on page 343.

RESTMAN resetprops

Re-initializes a REST Web application's runtime.props file to the current value of the REST Management Agent's default.props file.
Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman -name instance-name -appname application-name -resetprops</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
</tbody>
</table>

-name instance-name

The name of the REST Management Agent instance.

-appname application-name

The name of the REST Web application.

For information on the properties, see Reference to OpenEdge Web Service Properties on page 343.

**RESTMAN resetstats (Application)**

Resets the statistics for a REST Web application.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman -name instance-name -appname application-name -resetstats</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
</tbody>
</table>

-name instance-name

The name of the REST Management Agent instance.

-appname application-name

The name of the REST Web application.

For more information on these statistics, see the OpenEdge Management and OpenEdge Explorer: Configuration manual or the OpenEdge Management or OpenEdge Explorer online help.

**RESTMAN republish (Application)**

Redeploys only those files that you specify in the command to their corresponding locations in the Web application folder on a Web server.
You can add, update, or delete files from the REST Web application using the republish command.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman -name instance-name -appname application-name -filename [file-name.paar</td>
</tr>
<tr>
<td>Windows</td>
<td>restman -name instance-name -appname application-name -filename [file-name.paar</td>
</tr>
</tbody>
</table>

-name instance-name

The name of the REST Management Agent instance.

-appname application-name

The name of the REST Web application.

-filename [file-name.paar | application-name.zip ]

The name of the REST Web application's paar file or the name of the ZIP file (that contains all the files according to the folder structure of the REST Web application) that you want to republish.

For example, if you want to publish a new or an already deployed paar file of a Web application on the Web server, then you must provide the file-name.paar filename and extension in the syntax.

For other files in the REST Web application project, you must create a ZIP file with all the files that you want to republish, placed according to the folder structure of the REST Web application on the Web server, and then republish it using the republish command. For example, if you have a REST Web application, myApplication, on the Web server, and if you want to republish the Web application's security configuration file, appSecurity-anonymous.xml, at the Web server location apache-tomcat-instance/webapps/myApplication/WEB-INF, you must place the security file in a folder with the folder structure myApplication/WEB-INF/adapters/appSecurity-anonymous.xml, and compress it as myApplication.zip and specify myApplication.zip in the republish command.

You can also create the above ZIP file using the RESTGEN utility. For more information on RESTGEN utility, see RESTGEN on page 377.

Note: You must add the versionInfo.json file to your ZIP file if you want to delete any file from the REST Web application, or if you want to simultaneously update, add, and delete files using the RESTMAN republish command. For more information on creating and adding the versionInfo.json file to your ZIP file, see Deleting a REST Web application's files on page 391.

Notes

The republish command throws an error if you try to redeploy a REST Web application that does not have the same folder structure as that of the application in the Web server. If an error occurs while using republish, you can use the RESTMAN utility's deploy command to deploy a fresh instance of the REST Web application. For more information about deploying a fresh instance of a REST Web application, see RESTMAN deploy on page 381.
Deleting a REST Web application's files

You can also use the RESTMAN republish command to delete files from a REST Web application's instance on the Web server.

To delete specific files from the REST Web application, the ZIP file (whose name you specify in the RESTMAN republish command) must consist of a versionInfo.json file inside the WEB-INF directory that has the information about the files you want to delete. Therefore, you must create and add a versionInfo.json file in which you specify the files that you want to delete from the REST Web application. The following is a sample versionInfo.json file that lists REST Web application files to be deleted:

```
{
  "version": "03.18.2013:12:11:15"
  "deleted": ["MyApp/", "MyApp/static", "MyApp/static/auth",
              "MyApp/static/auth/error404.html"]
}
```

In this sample, the file, error404.html, is deleted from the REST Web application's instance on the Web server.

Although, versionInfo.json is not essential when publishing a new or an existing file on the Web server, if you want to simultaneously add, update, and delete files of your REST Web application, then you must include the relevant instructions in the versionInfo.json file as shown in the following sample:

```
{
  "version": "03.18.2013:12:11:15"
  "updated": ["MyApp/", "MyApp/WEB-INF/", "MyApp/WEB-INF/adapters",
              "MyApp/WEB-INF/adapters/MyApp.paar", "MyApp/WEB-INF/adapters/runtime.props"],
  "added": ["MyApp/", "MyApp/static", "MyApp/static/auth",
            "MyApp/static/auth/error402.html", "MyApp/static/auth/error500.html"],
  "deleted": ["MyApp/", "MyApp/static", "MyApp/static/auth",
              "MyApp/static/auth/error404.html"]
}
```

In this sample, the MyApp.paar and runtime.props files are updated, the error402.html and error500.html files are added, and the error404.html file is deleted from the REST Web application's instance on the Web server.

RESTMAN resetstats (Management Agent)

Resets the statistics for a REST Management Agent.
### Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td><code>restman</code></td>
</tr>
<tr>
<td></td>
<td><code>-name instance-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-resetstats</code></td>
</tr>
<tr>
<td>Windows</td>
<td><code>restman</code></td>
</tr>
<tr>
<td></td>
<td><code>-name instance-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-prop property-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-value new-property-value</code></td>
</tr>
<tr>
<td></td>
<td><code>-setdefaults</code></td>
</tr>
</tbody>
</table>

- `name instance-name`  
  The name of the REST Management Agent instance.

For more information on these statistics, see the OpenEdge Management and OpenEdge Explorer: Configuration manual or the OpenEdge Management or OpenEdge Explorer online help.

### RESTMAN setdefault (Management Agent)

Sets one of the properties of the Rest Management Agent's `default.props` to its default value.

All the properties of the Rest Management Agent that are used when initializing a REST Web application are in its `default.props` file.

#### Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td><code>restman</code></td>
</tr>
<tr>
<td></td>
<td><code>-name instance-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-prop property-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-value new-property-value</code></td>
</tr>
<tr>
<td></td>
<td><code>-setdefaults</code></td>
</tr>
</tbody>
</table>

- `name instance-name`  
  The name of the REST Management Agent instance.

- `prop property-name`  
  The property of the REST Management Agent.

- `value new-property-value`  
  The new value you want to set as default for the property.

For information on the properties, see Reference to OpenEdge Web Service Properties on page 343.
RESTMAN setprops (Application)

Sets a property of a REST Web application.

All the properties of an application are in its `runtime.props` file.

**Note:** You must not set the `serviceAvailable` property using the `setprops` command as that might lead to errors. You must set the property using RESTMAN `enable` and RESTMAN `disable`. For more information about the commands, see RESTMAN `enable` on page 382 and the RESTMAN `disable` on page 381.

### Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td><code>restman</code>&lt;br&gt;<code>-name instance-name</code>&lt;br&gt;<code>-appname application-name</code>&lt;br&gt;<code>-prop property-name</code>&lt;br&gt;<code>-value new-property-value</code>&lt;br&gt;<code>-setprops</code></td>
</tr>
</tbody>
</table>
| Windows          | -name instance-name  
|                  | The name of the REST Management Agent instance.  
|                  | -appname application-name  
|                  | The name of the REST Web application.  
|                  | -prop property-name  
|                  | The property of the REST Web Application.  
|                  | -value new-value-name  
|                  | The new value you want to set as default for the property.  

For information on the properties, see Reference to OpenEdge Web Service Properties on page 343.

RESTMAN setprops (Management Agent)

Sets one of the properties of a REST Management Agent.

All the properties of the REST Management Agent are in its `oerm.properties` file.
Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td><code>restman</code>&lt;br&gt;<code>-name instance-name</code>&lt;br&gt;<code>-prop property-name</code>&lt;br&gt;<code>-value new-property-value</code>&lt;br&gt;<code>-setprops</code></td>
</tr>
<tr>
<td>Windows</td>
<td><code>restman</code>&lt;br&gt;<code>-name instance-name</code>&lt;br&gt;<code>-prop property-name</code>&lt;br&gt;<code>-value new-property-value</code>&lt;br&gt;<code>-setprops</code></td>
</tr>
</tbody>
</table>

- **-name instance-name**
  
  The name of the REST Management Agent instance.

- **-prop property-name**
  
  The property of the REST Web application.

- **-value new-value-name**
  
  The new value you want to set as default for the property.

For information on the properties, see Reference to OpenEdge Web Service Properties on page 343.

### RESTMAN undeploy

Undeploys a REST Web application from a REST Management Agent.

Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td><code>restman</code>&lt;br&gt;<code>-name instance-name</code>&lt;br&gt;<code>-appname application-name</code>&lt;br&gt;<code>-undeploy</code></td>
</tr>
<tr>
<td>Windows</td>
<td><code>restman</code>&lt;br&gt;<code>-name instance-name</code>&lt;br&gt;<code>-appname application-name</code>&lt;br&gt;<code>-undeploy</code></td>
</tr>
</tbody>
</table>

- **-name instance-name**
  
  The name of the REST Management Agent instance.

- **-appname application-name**
  
  The name of the REST Web application.

### RESTMAN unpublish (Application)

Deletes a REST service (pair) from a REST Web Application.
**Note:** You must disable the REST Web application before you can unpublish a service from it.

### Syntax

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>restman</td>
</tr>
<tr>
<td>Windows</td>
<td>-name instance-name</td>
</tr>
<tr>
<td></td>
<td>-appname application-name</td>
</tr>
<tr>
<td></td>
<td>-service service-name</td>
</tr>
<tr>
<td></td>
<td>-unpublish</td>
</tr>
</tbody>
</table>

- **-name instance-name**
  
  The name of the REST Management Agent instance.

- **-appname application-name**
  
  The name of the REST Web application.

- **-service service-name**
  
  The name of the REST Web application’s .paar file that you want to unpublish. You must provide only the service name, and not the complete filename and extension.

  For example, you must provide `myService` as your service name, instead of `myService.paa`.

### Notes

The **unpublish** command throws an error if you try to undeploy a web service of REST Web application that has only one service deployed on the Web server. Therefore, a REST Web application must have at least two Web services (.paar files) deployed on the Web server for you to be able to unpublish one of the available services of the REST Web application.

---

**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>Windows</td>
<td></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>UNIX</td>
<td><code>wsaman -name wsinstance-name -wsm wsm-location-on-wsaman-machine</code></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 the following table, which lists the WSDL style/use combinations supported.

  **Table 104: 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 Management/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</td>
<td>wsaman -name wsainstance-name {-appname app-friendlyname</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.

Notes

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>UNIX</td>
<td><code>wsaman -name wsinstance-name</code></td>
</tr>
</tbody>
</table>
| Windows          | `{ -appname app-friendyname
|                 |   -namespace app-targetnamespace }` |
|                  | -enable |

- `name wsinstance-name`  
  The name of a WSA instance.

- `appname app-friendyname`  
  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</td>
<td><code>wsaman</code> <code>-name wsainstance-name</code> <code>{ </code>-appname app-friendlyname`</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.

- **-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.

**Notes**

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</td>
<td><code>wsaman -name wsainstance-name -getdefaults</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 properties, see Reference to OpenEdge Web Service Properties on page 343.

## 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>UNIX</td>
<td>`wsaman -name wsainstance-name {-appname app-friendlname</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-friendlname**
  
  The friendly name of the Web service.

- **-namespace app-targetnamespace**
  
  The target namespace of the Web service.

For more information on these properties, see Reference to OpenEdge Web Service Properties on page 343.

### Notes

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
• enableWsd1
• enableWsd1Listings
• 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>UNIX</td>
<td><code>wsaman</code></td>
</tr>
<tr>
<td></td>
<td><code>-name wsainstance-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-getprops</code></td>
</tr>
<tr>
<td>Windows</td>
<td><code>wsaman</code></td>
</tr>
<tr>
<td></td>
<td><code>-name wsainstance-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-getprops</code></td>
</tr>
</tbody>
</table>

-name wsainstance-name

The name of a WSA instance.

Notes

The values of these properties can be temporarily overridden through WSAMAN setprops (WSA) on page 410. Such changes are not written to ubroker.properties. If there are such changes in effect, the values displayed by WSAMAN getprops (WSA) on page 402 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>
<tbody>
<tr>
<td>UNIX</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wsaman</td>
</tr>
<tr>
<td></td>
<td>-name</td>
</tr>
<tr>
<td></td>
<td>wsainstance-name</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>-appname</td>
</tr>
<tr>
<td></td>
<td>app-friendlyname</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>app-targetnamespace</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>-getstats</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.

For more information on these statistics, see the OpenEdge Management or 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>
<tbody>
<tr>
<td>UNIX</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wsaman</td>
</tr>
<tr>
<td></td>
<td>-name</td>
</tr>
<tr>
<td></td>
<td>wsainstance-name</td>
</tr>
<tr>
<td></td>
<td>-getstats</td>
</tr>
</tbody>
</table>

- `name wsainstance-name`

  The name of a WSA instance.

For more information on these statistics, see the OpenEdge Management or 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>Windows</td>
<td></td>
</tr>
</tbody>
</table>

- **-name wsinstance-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 the following table, which lists the WSDL style/use combinations supported.

  **Table 105: 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 Management or 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></td>
</tr>
</tbody>
</table>

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>
</table>
| UNIX             | `wsaman`<br>`-name wsainstance-name`<br>`{-appname app-friendlyname`<br>`| `-namespace app-targetnamespace` `}
| Windows          | `-query` |

- `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</td>
<td><code>wsaman</code>&lt;br&gt;<code>-name wsainstance-name</code>&lt;br&gt;<code>-query</code></td>
</tr>
<tr>
<td>Windows</td>
<td><code>-query</code></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>UNIX</td>
<td>wsaman -name wsainstance-name -resetdefaults</td>
</tr>
<tr>
<td>Windows</td>
<td>-name wsainstance-name</td>
</tr>
</tbody>
</table>

- name wsainstance-name

The name of a WSA instance.

For more information on these properties, see Reference to OpenEdge Web Service Properties on page 343.

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>
<tbody>
<tr>
<td>UNIX</td>
<td>wsaman -name wsainstance-name</td>
</tr>
<tr>
<td>Windows</td>
<td>-name wsainstance-name</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.
The target namespace of the Web service.

For more information on these properties, see Reference to OpenEdge Web Service Properties on page 343.

**Notes**

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>
<tbody>
<tr>
<td>UNIX</td>
<td><code>wsaman -name wsinstance-name</code></td>
</tr>
</tbody>
</table>
| Windows          | `-{ -appname app-friendyname`  
|                  | `| -namespace app-targetnamespace } -resetstats`                                           |

- **-name wsinstance-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 statistics, see the OpenEdge Management or OpenEdge Explorer online help for Web services.

---

### WSAMAN resetstats (WSA)

Resets the statistics for a WSA instance.
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</td>
<td><code>wsaman</code></td>
</tr>
<tr>
<td></td>
<td><code>-name wsainstance-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-resetstats</code></td>
</tr>
<tr>
<td>Windows</td>
<td><code>wsaman</code></td>
</tr>
<tr>
<td></td>
<td><code>-name wsainstance-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-prop property-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-value property-value</code></td>
</tr>
<tr>
<td></td>
<td><code>-setdefaults</code></td>
</tr>
</tbody>
</table>

=name wsainstance-name

The name of a WSA instance.

For more information on these statistics, see the OpenEdge Management or OpenEdge Explorer online help for Web services.

Note: This parameter is case sensitive.

=value property-value

The value of the property.

For more information on these properties, see the Reference to OpenEdge Web Service Properties on page 343.
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>UNIX</td>
<td><code>wsaman -name wsainstance-name</code></td>
</tr>
<tr>
<td></td>
<td>`{ -appname app-friendlyname</td>
</tr>
<tr>
<td></td>
<td><code>-prop property-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-value property-value</code></td>
</tr>
<tr>
<td></td>
<td><code>-setprops</code></td>
</tr>
<tr>
<td>Windows</td>
<td><code>-name wsainstance-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-appname app-friendlyname</code></td>
</tr>
<tr>
<td></td>
<td><code>-namespace app-targetnamespace</code></td>
</tr>
<tr>
<td></td>
<td><code>-prop property-name</code></td>
</tr>
<tr>
<td></td>
<td><code>-value property-value</code></td>
</tr>
<tr>
<td></td>
<td><code>-setprops</code></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 Reference to OpenEdge Web Service Properties on page 343.

**Notes**

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>wsaman -name wsinstance-name -prop property-name -value property-value -setprops</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
</tr>
</tbody>
</table>

- **-name wsinstance-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 Java container 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>
</table>
| UNIX Windows     | `wsaman -name wsainstance-name`<br>`{ -appname app-friendlyname`|<br>`| -namespace app-targetnamespace }`<br>`-undeploy`<br>`-name wsainstance-name`<br>`-appname app-friendlyname`<br>`-namespace app-targetnamespace`<br>`Notes`<br>When you undeploy a Web service, whether you use the `WSAMAN` utility or OpenEdge Management/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`<br>`-wsm wsm-location-on-wsaman-machine`<br>`{ -appname app-friendlyname`|<br>`| -namespace app-targetnamespace }`<br>`[-encoding encodingstyle ]`<br>`-update`<br>`-name wsainstance-name`<br>`-appname app-friendlyname`<br>`-namespace app-targetnamespace`<br>`-encoding encodingstyle`<br>`update`<br>`-name wsainstance-name`<br>`-appname app-friendlyname`<br>`-namespace app-targetnamespace`<br>`encodingstyle`<br>`update`<br>`-name wsainstance-name`<br>`-appname app-friendlyname`<br>`-namespace app-targetnamespace`<br>`encodingstyle`<br>`update`
-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 the following table, which lists the WSDL style/use combinations supported.

**Table 106: Setting the SOAP format for update 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**

- 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 WSAMAN update on page 412 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>
</table>
| UNIX             | `wsconfig` [  
| Windows          | `[ [ -name component-name]`  
|                  | [ -propfile path-to-properties-file]  
|                  | [ -messenger ]  
|                  | [ -validate ]  
|                  | ] | -help ] |

- **-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.

### Example

The following table shows several examples that use the **WSCONFIG** command. Assume the Transaction Server name is **wsbroker1**.
**Table 107: WSCONFIG command examples**

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

**WTBMAN**

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>&lt;br&gt;<code>  { -name transaction-server-name</code>&lt;br&gt;`  { -kill</td>
</tr>
</tbody>
</table>
-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.

Example

The following table 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 108: 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&lt;sup&gt;20&lt;/sup&gt;</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&lt;sup&gt;20&lt;/sup&gt;</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&lt;sup&gt;20&lt;/sup&gt;</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&lt;sup&gt;20&lt;/sup&gt;</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&lt;sup&gt;20&lt;/sup&gt;</td>
<td>wtbman -name wsbroker1 -host finance -port 9999 -user tom -stop</td>
</tr>
</tbody>
</table>

### Notes

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`.

---

<sup>20</sup> Prompts for a password.
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.

- `agentdetail`

  Displays the procedure run by a WebSpeed agent. You can select the specific WebSpeed agent by specifying the process ID (PID) of the agent process.

  **Example**

  Use the following command to display the `.r` procedure run by a WebSpeed agent. It displays the port and connection state of a WebSpeed agent.

  ```shell
  wtbman -name <name> -agentdetail <pid>
  ```

- `agentkill`

  Stops a specific WebSpeed agent forcefully. The PID specified to the `wtbman` command must be an agent process for the specified WebSpeed. Use the `kill` command to stop a specific WebSpeed agent on UNIX. Use the `taskkill` command to stop a specific WebSpeed agent on Windows XP Professional Edition. This utility is not supported by the Windows Home Edition.

  **Example**

  Use the following command to stop a WebSpeed agent forcefully.

  ```shell
  wtbman -name <name> -agentkill <pid>
  ```

- `refresh`

  Allows the users to trim individual agents and restart new ones in order to gradually migrate to a new version of r-code from their current version. Details of a Refresh operation are logged at a verbose level.

  **Example**

  Use the following command to refresh an AppServer.

  ```shell
  wtbman -name <name> -refresh
  ```

- `cancelRefresh`

  Cancels a refresh command. This command cancels a refresh operation that is running for too long. It will also stop any busy agent marked for refresh from getting trimmed automatically. The broker will not add new agents for any agent that were trimmed when the cancelled refresh operation was running. An administrator can add more agents if needed. After the refresh is cancelled, a message is returned indicating that the refresh was successful. If no refresh was active for that broker, the following message is returned:

  Refresh for `<brokerName>` is not active
Example

Use the following command to cancel the refresh command issued for an AppServer.

```
wtbman -name <name> -cancelRefresh
```

- refreshStatus

This command returns detailed information that is captured in the following table:

**Table 109: Refresh detail view output fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refresh Active</td>
<td>Yes/No</td>
<td>If the refresh is not active, and if there was a refresh operation issued earlier, it shows the refreshStatus of that that operation</td>
</tr>
<tr>
<td>Refresh Start Time</td>
<td>yyyy-MM-dd'T'HH:mm:ss.SSS</td>
<td>Start time of the currently executing refresh operation. If no refresh is active, it shows the start time of the latest refresh operation that was completed</td>
</tr>
<tr>
<td>Refresh End Time</td>
<td>yyyy-MM-dd'T'HH:mm:ss.SSS</td>
<td>End time of the latest refresh operation that was completed. This is not applicable for a running refresh operation</td>
</tr>
<tr>
<td>Refresh Cancel Time</td>
<td>yyyy-MM-dd'T'HH:mm:ss.SSS</td>
<td>The time when a refresh operation was cancelled</td>
</tr>
<tr>
<td>Agents at refresh start</td>
<td>Integer value &gt;= 0</td>
<td>Number of agents that were running at the time when refresh command was issued</td>
</tr>
<tr>
<td>Agents marked for refresh</td>
<td>Integer value &gt;= 0</td>
<td>Number of agents that are marked to be refreshed. Agents that are busy will be marked for refresh and will be stopped only after they become available. Also the agents that were added after refresh operation started will not be refreshed. This field will be displayed for active refresh operation only.</td>
</tr>
<tr>
<td>New agents added during refresh</td>
<td>Integer value &gt;= 0</td>
<td>No. of agents that got added after the refresh was issued. These agents won’t be running old r-code, so they will not be marked for refresh</td>
</tr>
<tr>
<td>Field</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Agents at refresh cancel</td>
<td>Integer value &gt;= 0</td>
<td>No. of agents running at the time of refresh cancellation</td>
</tr>
<tr>
<td>Agents refreshed</td>
<td>Integer value &gt;= 0</td>
<td>It shows the actual number of agents refreshed for a refresh operation that is complete</td>
</tr>
<tr>
<td>Refresh operation count</td>
<td>Integer value &gt;= 0</td>
<td>No. of times a refresh operation is invoked including the currently executing refresh operation since start of the broker</td>
</tr>
<tr>
<td>Refresh cancel count</td>
<td>Integer value &gt;= 0</td>
<td>No. of times a refresh cancellation was is invoked since start of the broker</td>
</tr>
<tr>
<td>PID of agents marked for refresh</td>
<td>Integer value &gt;= 0</td>
<td>Number of agents added after the refresh was issued. This doesn't include the agents that were started as replacement agents. This include agents added due to addservers or new client requests.</td>
</tr>
<tr>
<td>PID of agents refreshed</td>
<td>Integer value &gt;= 0</td>
<td>List the PID of agents that were Refreshed(replaced by new agents).</td>
</tr>
</tbody>
</table>

**Example**

Use the following command to query the status of a refresh command issued for an AppServer.

```
wtbman -name <name> -refreshStatus
```
Reference to Dynamic Server Properties

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.

For details, see the following topics:

- 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. The following table describes the valid settings for this property:

### Table 110: `allowRuntimeUpdates` property values

<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 <code>ubroker.properties</code> file and reload all value and behavioral properties. This is true whether the file is directly edited or changed through OpenEdge Management/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`
- `agentDetailTimeout`

### 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`
Dynamic properties

- srvrMaxPort
- srvrMinPort
- srvrStartupParam
- 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
OERealm

OERealm is a component of the OpenEdge Single Point of Authentication architecture where the authentication system of various applications use a single user account information source. It provides user account information, maintained in an AppServer, to the login process of an external product’s authentication system. OERealm has three components – OERealm Client, OERealm Server OOABL interface, and OERealm server class. The OERealm client uses abstract user account property names and the OERealm server class supplies the mapping between abstract and actual user account property names. The OERealm server class runs in a session-free AppServer as a singleton OOABL class because session-free supports multi-threaded OERealm clients and fail-over. OERealm server class uses user account numerical IDs for unique user IDs and it manages identification information for multiple OpenEdge domains or tenants. OERealm supports the following Progress products:

- Progress OpenEdge BPM
- Progress OpenEdge Rest services and for Progress OpenEdge
- Progress Rollbase

OERealm is not an authentication system in itself and it does not support the control the success or failure of a user login through a remote authentication system. It can use user tables other than those of OpenEdge database and supports more than one OpenEdge domain or tenant. It allows the implementation of multiple classes per AppServer and also functions with non-OpenEdge OERealm clients.

**Note:** OERealm does not support remotely created and deleted user accounts.

For details, see the following topics:

- Implementing OERealm
- Supported OERealm Clients
Implementing OERealm

The OERealm server class runs in a Session-Free AppServer as a Singleton OOABL class. To implement the OERealm Server Class, do the following:

1. Implement the IHybridRealm Interface
2. Implement the necessary methods
3. Implement security in OERealm Class (Optional)

Implement the IHybridRealm Interface

Create a singleton OOABL class that implements the OERealm HybridRealm interface. A sample interface implementation class, HybridRealm.cls, can be found at $DLC/src/samples/security/OpenEdge/security/Realm if you have the OpenEdge Documentation and Samples installed.

Implementing necessary methods

Implement the following methods:

Table 111: Methods defined in the HybridRealm.cls class

<table>
<thead>
<tr>
<th>Method</th>
<th>Mandatory?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValidateUser</td>
<td>Yes</td>
<td>This method searches for the user account, validates whether the account domain is valid and enabled, and returns the account numeric ID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The OERealm service interface uses a numerical user account ID as an ABL integer instead of the full user account name. If the user account storage of your OERealm service does not support the integer user account IDs, it is recommended that you build a mechanism to permanently assign unique ABL integer values to each user account in the storage.</td>
</tr>
<tr>
<td>ValidatePassword</td>
<td>Yes</td>
<td>This method validates the client-supplied password of the user account against the stored password.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> You can define both the clear-text and digest form of the ValidatePassword() method.</td>
</tr>
<tr>
<td>GetAttribute</td>
<td>Yes</td>
<td>This method retrieves an attribute for the specified user account.</td>
</tr>
</tbody>
</table>
### Implementing security in OE Realm Class (optional)

Users can implement an extra layer of security in the OERealm Server class. To implement this level of security, each request from an OERealm Client should contain a valid and sealed Client-Principal (C-P).

1. Use the `genpassword` utility to generate a Client Principal (C-P) token file and distribute it to the intended OERealm clients.

2. Configure this C-P token as the Realm token file in the OERealm Client configuration to ensure that it is passed along with each appserver request that is sent to the OERealm Client. Refer to Configuring OERealm Clients on page 428 for details on how different OERealm clients can be configured.

3. In the OERealm Server class methods, allow the request to proceed only if the AppServer request contains a valid and sealed C-P.

### Supported OERealm Clients

OERealm supports the following clients for authentication purposes:

- Progress OpenEdge BPM
- Progress OpenEdge REST services and for Progress OpenEdge
- Progress Rollbase
Configuring OERealm Clients
Clients that connect to and communicate with OERealm should be configured with the following values:

**Table 112: OERealm client configuration details**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Configuration attribute and value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Realm URL</td>
<td>A valid URL to connect to the OERealm</td>
</tr>
<tr>
<td>Realm Class</td>
<td>The fully qualified name of the OERealm Server class.</td>
</tr>
<tr>
<td>Realm Token File</td>
<td>if OERealm is configured to require each request to contain a valid C-P, the client must have access to the C-P file generated by the implementer of OERealm.</td>
</tr>
<tr>
<td>Realm Attribute names</td>
<td>The clients must configure attribute names that the OERealm server class can use to get an attribute.</td>
</tr>
</tbody>
</table>

**Note:** Realm URL can take the following values:

- "AppServer[s]://<name-server-host>:<name-server-port>/<service-name>" for remote OpenEdge AppServer
- "AppServerDC[s]://<AppServer-host>:<AppServer-port>/<service-name>" for remote OpenEdge AppServer using Direct Connect
- http[s]://<host>[:<port>]/[oeabl-app-name]/apsv" for remote PAS for OpenEdge server
- "http[s]://<host>[:<port>]/<aia-app-name>/aia" for remote classic AppServer using AppServer Internet Adapter
- "internal://localhost/nxgas" for local PAS for OE Server

The following table provides the details for various OERealm supported clients and their configuration details:

**Table 113: OERealm client configuration details**

<table>
<thead>
<tr>
<th>Client</th>
<th>Configuration attribute and value</th>
</tr>
</thead>
</table>
| Progress OpenEdge BPM| Configure following attributes in BPSHOME/conf/umacl.conf  
|                     | oeauth.server.location = <Realm URL>  
|                     | oeauth.server.provider = <Fully qualified OO ABL Server Class>  
|                     | oeauth.server.authmethod = [basic|digest]  
|                     | oeauth.server.clientprincipal = <realm Token file>  
|                     | For more information, refer to Progress OpenEdge Business Process Server: Business Process Portal Administrator’s Guide |
| Progress Rollbase   | OpenEdge Realm URL  
|                     | OpenEdge Realm Class  
|                     | Realm Token File  
|                     | For more information refer to Rollbase documentation.  

---

OpenEdge Application Server: Administration 429
<table>
<thead>
<tr>
<th>Client</th>
<th>Configuration attribute and value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST</td>
<td>realmURL</td>
</tr>
<tr>
<td></td>
<td>realmClass</td>
</tr>
<tr>
<td></td>
<td>realmTokenFile</td>
</tr>
<tr>
<td></td>
<td>For more details see OERealmUserDetails properties on page 333</td>
</tr>
</tbody>
</table>

### OERealm Best Practices

The OERealm client treats any errors or exceptions as fatal server errors and different OERealm clients handle errors and exceptions differently.

The best practices for users of OERealm are as follows:

- No errors or exception should be sent from the `validateUser()` method.
- Errors should be sent from other methods when the user account number does not correspond to a physical user account or if the library used to access the physical user accounts returns an exception.
- All errors and warnings should be logged based on User Account Name or user-account-number.
- Return the defined defaults for accessing user account properties when the account property does not exist or the account property is defined as *Null* or *Unknown*.

### OERealm Security Best Practices

are as follows:

- Use OS file system security to protect your OERealm ABL code modules.
- Use SSL/TLS for non-local network connections.
- If possible, provide a dedicated AppServer for the OERealm functionality.
- Authenticate and authorize each request using the OERealm client's Client-Principal.
- Provide adequate capability to handle ABL errors and exceptions.

### Samples

If you have the OpenEdge Documentation and Samples installed, you can find a sample interface implementation class, `HybridRealm.cls`, at `$DLC\src\samples\security`. The sample contains README.txt that describes the process of creating an OpenEdge database with user account that will work with the sample OOABL server class.

You can access the sample files from the `openedge_doc` directory, contained in the `.zip` or `.tar` files available on the Progress ESD (Electronic Software Download) site or from [http://communities.progress.com/pcom/docs/DOC-16074](http://communities.progress.com/pcom/docs/DOC-16074).
Index

A

Acceptor, HTTP/SOAP
  adding to SonicMQ broker 253
Active Server Page Messenger 223
adapconfig tool
  for BrokerConnect configuration 236
ADAPTCMG utility 359
adaptman tool
  for BrokerConnect administration 237
ADAPTMAN utility
example 360, 362, 370, 372, 374, 376, 395, 413, 415
Administration framework
AppServer 47
  preparing to use 51
  WebSpeed in Windows 149
  WebSpeed on UNIX 171
Administrative tasks
  configuring AppServer components 51, 79
  setting up the execution environment 57
  starting and managing AppServers
    UNIX requirements 51, 80
    using command-line utilities 60
    using OpenEdge Management or OpenEdge Explorer 59
    Windows requirements 51
summary 71
AdminServer
  defined
    WebSpeed in Windows 149
    WebSpeed on UNIX 171
  definition
    AppServer 47
PROADSV utility 376
starting in Windows 71, 158
starting on UNIX 71, 181, 376
starting with PROADSV utility 185
Advanced Features
  WebSpeed broker setting 150
  WebSpeed setting 150
Agent (continued)
  error messages 204
  international configurations 203
  language 204
  LIMBO status
    WebSpeed in Windows 165
    WebSpeed on UNIX 185
  LOCKED status
    WebSpeed in Windows 165
    WebSpeed on UNIX 185
  starting dynamically
    WebSpeed in Windows 167
    WebSpeed on UNIX 187
  STARTING status
    WebSpeed in Windows 165
    WebSpeed on UNIX 185
  trimming running
    WebSpeed in Windows 167
    WebSpeed on UNIX 188
  Agent application mode 208
AIA
  Advanced features 80
  AIACONFIG Utility 361
  connection timeout 80
  SSL 80
  AIACONFIG utility 361
  Application Internet Adapter
    AIACONFIG utility 361
  Application PROPATH
    AppServer setting 52, 150
  Application Service
    assigning names
      using OpenEdge Management or OpenEdge Explorer 52, 150
    overview 40
  Application Service names
    AppServer setting 52, 150
  Applications 277
AppServer
  administration framework 47
  ASBMAN utility 362
  ASCONFIG utility 370
  code-page management 62
  component distribution 40
  configuring
    using OpenEdge Management or OpenEdge Explorer 80
  configuring the components 51, 79
  fault-tolerant instances 43
  fault-tolerant NameServers 43
AppServer (continued)
load balancing, overview 43
operating modes
overview 44
overview 32
registration overview 46
run-time components 40
session startup parameters 46
starting and managing 59
startup and shutdown, overview 46
status checking 61, 71
AppServer agent
administration 50
customizing the executable 58
definition 40
AppServer broker
definition 40
AppServer execution
preparing the environment 57
AppServer Internet Adapter (AIA)
configuration information 84
configuring 79
connection status 83
general administration 76
installation 76
instance name 77
overview 32
testing configuration 77, 83
test utilities 83, 426–428, 430
appServerKeepAlive service property 348
appServiceConnectionMode 348
appServiceHost service property 348
appServiceName service property 348
appServicePort service property 348
appServiceProtocol service property 348
architecture 270
ASBMAN utility
syntax 60
using 60
ASCONFIG utility
example 56, 105
syntax 56, 105
using 56
ASP example 225
ASP Web page 224
Audio files (.au)
stored on Web server
UNIX 178
Windows 157
Authentication
WSA 91
Authorization
WSA 91
Auto startup
AppServer setting 52, 150
AVAILABLE status 61

B

BrokerConnect
adaptoconfig tool 359
ADAPTCONFIG utility 359
ADAPTCONFIG Utility 359
adaptman tool 237, 360
ADAPTMAN utility 360
ADAPTMAN Utility 360
and NameServer 233
attributes specified in ubroker.properties 234
Command line configuration 236
Configuring 233
connection timeout 80
HTTP and HTTPS tunneling 231
security 231
SSL 80, 230
ubroker.properties 234
BUSY status 61

C

CGI Messenger
 configuring on UNIX 176
 executable
 UNIX 173
 Windows 154
 invoking on UNIX 176
Client application
definition 40
Client applications
administration 48
Client port range 220
clientASKActivityTimeout service property 349
clientASKResponseTimeout service property 349
ClientConnect
 Configuring 232
 SMQConnect 232
Code page
 used by JMS client 239
Code pages
 client and AppServer 62
 run-time character conversions 63
 settings 63
Command line tools
adaptconfig 359
adaptman 237, 360
Command-line utilities
 NSCONFIG
 UNIX 172
 Windows 154
 NSMAN
 UNIX 172
 Windows 153
summary 71
Command-line utilities (continued)
  WSCONFIG
    UNIX 172
    Windows 153
  WTBMAN
    UNIX 172
    Windows 153
Component distribution 42
Configuration information
  for AIA 84
Configurations
  Messenger-only installs
    Windows 164
  UNIX overview 170
  validating with status.p
    UNIX 187
    Windows 166
Windows overview 148
Configuring
  AIA 79
  AppServers
    in ubroker.properties file 50, 55, 82
    using OpenEdge Management or OpenEdge Explorer 80
  NameServers
    using OpenEdge Management or OpenEdge Explorer 51
  WebSpeed
    using OpenEdge Management or OpenEdge Explorer 150
  WSA for Web services 106, 275
  WSA instances 103, 274
CONNECT statement 198
Connection status
  for AIA 83
Connection-level fault tolerance, definition 40
  connectionLifetime service property 349
  connectionTimeout service property 349
Controlling NameServer
  AppServer setting 52, 150
  definition 40
Creating
  Session objects 232
  WSA instances
    multiple 102
Customizing the AppServer agent executable 58

D

Database
  code page 204
    connecting locally 198
  connecting programmatically 198
  connecting remotely 199
  multi-national 201
  shared memory connections 198
  Database (continued)
    Sports2000 197
Database server
  starting 197
DataServers
  and WebSpeed architecture 200
    connecting with 199
  local 200
    managing agent connections 201
    remote 200
Date (-d) startup parameter 204
DB2 database
  connecting to 199
DB2/400 database
  connecting to 199
Debug mode 211
Debugger enabling
  AppServer setting 52
  WebSpeed setting 150
Default service
  AppServer setting 52
  WebSpeed setting 150
deploy function
  WSA for Web services 106, 275
Deployment directory
  changing for Web services 111
Deployment files
  OpenEdge Web service 110
  Web service management 113
Deployment information
  for Sonic ESB services 246–247
Development mode
  WebSpeed security 207
Digital certificates
  and Web servers 79
    installing for WSA 91, 287
Directory
  changing for Web service deployment 111
disable function
  WSA for Web services 106, 275
DLC environment variable
  setting for NSAPI Web server
    UNIX 175
Document root directory
  static HTML files in
    UNIX 178
    Windows 157
Dynamic code-page support 191

E

enable function
  WSA for Web services 106, 275
Enabling
  OpenEdge Web services 112
Endpoints

SonicMQ 248

Environment variables

BrokerConnect setting 233
for Java container 77
NameServer setting 52, 150
PROMSGS 202

PROPATH
Windows 158
where to set 57

WRKDIR
UNIX 182
Windows 159

ESBOEGEN utility 371
European Numeric Format (-E) startup parameter 204

export function
WSAMAN utility 399

Exporting
Web services
Importing
Web services 114

F

Fault-tolerant AppServers
overview 43

Fault-tolerant NameServers
overview 43

file upload directory 213

Files
log 58
ubroker.properties 50

Firewalls 217

G

GENUUID utility
UNIX 179

getdefaults function
WSAMAN utility 383, 400

getprops (Service) function
WSAMAN utility 401

getprops (WSA) function
WSAMAN utility 402

getstats (Service) function
WSAMAN utility 384, 386, 402

getstats (WSA) function
WSAMAN utility 403

H

Host Name (-H) startup parameter
database connections 198
remote database connections 199

HTTP/SOAP acceptor
adding to SonicMQ broker 253

HTTPS 215

HTTPS protocol
enabling for WSA 91, 287
for WSA 91, 287

I

idleSessionTimeout service property 350

IHybridRealm Interface
using with uri 426

IHybridRealm Interface implementation
using a Java container 426

Image files (.gif)
stored on Web server
UNIX 178
Windows 157

Implementing necessary methods
testing utilities 426

Implementing OERealm
using a Java container 426
using with uri 426

import function
WSAMAN utility 403

Informix database
connecting to 199

Initial servers to start 62

Initialization File (-ininame) startup parameter 204

initialSessions service property 350

Installation
OpenEdge Adapter for Sonic ESB 242
OpenEdge Adapter for SonicMQ 231

Instance name
for AIA 77

Internal Code Page (-cpinternal) startup parameter 204

Internationalization
code-page management 62

OpenEdge Adapter for SonicMQ 239

Internet Protocol List 210

ISAPI Messenger
executable
Windows 154

J

Java
class files location
UNIX 178
Windows 157

Java container
configurations with WSA 95, 271
configuring to recognize the WSA 90
large Web service messages 91
role in WSA security 117
using with uri 83, 426–428, 430
Java container (continued)
    virtual paths 77
Java Message Service (JMS)
    MapMessage 238
    performance considerations 237–238
    reusing messages 238
    StreamMessage 238
    TextMessage 238
Java virtual machine (JVM)
    memory management 91
JMS messaging
    calls 238
    Discardable messages 238
    Internationalization 239
    Load balancing 238
    Message reuse 238
    performance 237

Messenger (continued)
    managing
        UNIX 188
        Windows 168
    operating system compatibility
        UNIX 173
        Windows 155
    overview 33
    script file location
        UNIX 178
        Windows 157
Microsoft IIS
    Messenger executable
        Windows 154
Minimum servers kept running 62
msgSessions service property 351
Multi-user database 197

L
LIMBO status 61
list function
    WSAMAN utility 386, 405
Load balancing
    NameServer
        UNIX 182
        Windows 161
    overview 43
LOCKED status 61
Log files
    configuring
        UNIX 182
        Windows 159
    maintaining
        UNIX 184
        Windows 164
Logging properties
    AIA broker setting 80
    AppServer agent process setting 52
    AppServer broker setting 52
    WebSpeed agent process setting 150
    WebSpeed broker setting 150

M
Maximum servers kept running 62
maxSessions service property 350
Messaging
    ServerConnect 52, 150
Messenger
    downloading executables
        UNIX 173
        Windows 155
    executable location in Windows 154
    executable location on UNIX 173

N
n-tier deployment
    Messenger-only installs
        Windows 164
NameServer
    administration 49
    client port range 220
    configuring
        using OpenEdge Management or OpenEdge Explorer 51, 80
    controlling AIA instance 80
    controlling, overview 40
    fault tolerance 43
    load balancing
        UNIX 182
        Windows 161
    neighbors, overview 43
    NSCONFIG utility 372
    NSMAN utility 374
    overview 40
    predefined instance
        Windows 149, 171
    replication, overview 43
    starting
        Windows 162
    validating configurations
        UNIX 179
NAT 218
Native Invocation
    OpenEdge Adapter for Sonic ESB 241
Neighbor NameServers
    overview 43
Netscape
    Messenger executable
        UNIX 173
        Windows 154
    Network Address Translation 218
openedge adapter for sonic esb (continued)
security 256
service instance 248
service instance, creating 248
service properties 246, 253
ssl 257
using 245
web service invocation 241
openedge adapter for sonicmq
brokerconnect 230
client connections 230
clientconnect 230
code-page encoding 239
command line tools 236
configuring 231
configuring brokerconnect 233–234, 236
configuring clientconnect 232
configuring serverconnect 232
environment variables for 233
installation 231
internationalization 239
introducing 229
jms message size 238
maximizing performance 237
overview 33
reusing messages 238
server startup parameters 233, 236
serverconnect 230
openedge management or openedge explorer
brokerconnect configuration 233
configuring
appservers 80
nameservers 51, 80
described
windows 149, 171
starting and managing appservers 59
using to edit configuration
unix 179
windows 157
using with dataservers 200
validating configurations
unix 185
windows 165
openedge rdbms
connecting in webspeed 197
openedge service instance 248
openedge startup parameters 46
openedge web service
deploying 109, 278
properties
setting 347
summary overview 343
properties reference 343
openedge web services
administration scenarios 115

object.conf file
modifying
unix 174
sample
unix 174
oerelalm
for aia 426
testing utilities 426
openedge adapter for sonic esb
automatic upgrade of openedge containers 35
configuring the runtime container 34
default service properties 247
editing an instances of a service 253
editing service properties 246–247
exposing service 253
installation 242
installing 242
introducing 241
native invocation 241
and web services 253
services, sonic.esb; sonic esb 246
overview 34
resource editor 251
OpenEdge Web services (continued)
changing deployment directory 111
default properties 108, 278
development 107
deployment file management 113
enabling 112
exporting and importing 114
managing post-deployment 113, 281
monitoring and tuning 114, 280
post deployment 110
versioning 110
Operating Mode
AppServer setting 52, 150
Operating modes
overview 44
ORACLE database
connecting to 199
Owner Info
AppServer setting 52
WebSpeed setting 150

P
Parameter file
starting database with 197
passwords 214
Performance
JMS messaging 237
Port Number
AppServer setting 52, 150
port numbers 212
Ports
TCP 217
Post-deployment
Web service management 113, 281
priorityWeight property
UNIX 182
Windows 161
PROADSV utility
starting AdminServer
UNIX 171, 185
PROCFS environment variable
setting for NSAPI Web server
UNIX 175
Production mode
WebSpeed security 207
PROMSGS environment variable
international configurations 202
limits on NSAPI Web server 203
setting for NSAPI Web server
UNIX 175
PROPATH environment variable
overriding 52, 150
Windows 158
Properties
default Web service 108, 278
Properties (continued)
modifying WSA 105
reference for services 343
Web services deployment file 110
Properties file
editing and validating 55
guidelines for editing 55, 235
hierarchy 55, 235
Property file
overview
UNIX 179
Windows 157
ptpsession.p 232
Q
query (Service) function
WSAMAN utility 387, 405
query (WSA) function
WSAMAN utility 406
R
R-code for SmartDataObjects 58
Registration mode 218
Replicated NameServers, overview 43
requestWaitTimeout service property 352
resetdefaults function
WSAMAN utility 388, 407
resetprops function
WSAMAN utility 388, 407
resetstats (Service) function
WSAMAN utility 389, 391, 408
resetstats (WSA) function
WSAMAN utility 408
REST
Architecture 270
REST Management Agent Administration
Installing and Configuring 261
Managing
RESTMAN functions for managing 304
Security Configurations 276
REST Web application Administration
Managing
RESTMAN functions for managing 305
Security Configurations 296
RESTMAN deploy utility 381
RESTMAN disable utility 381
RESTMAN enable utility 382
RESTMAN getdefaults utility 383
RESTMAN getprops (Application) utility 383
RESTMAN getprops (Management Agent) utility 384
RESTMAN getstats (Application) utility 384
RESTMAN getstats (Management Agent) utility 386
RESTMAN list utility 386
RESTMAN query (Application) utility 387
RESTMAN query (Management Agent) utility 387
RESTMAN republish (Application) utility 389
RESTMAN resetdefaults (Management Agent) utility 388
RESTMAN resetprops utility 388
RESTMAN resetstats (Management Agent) utility 391
RESTMAN setdefault (Management Agent) utility 392
RESTMAN setprops (Application) utility 393
RESTMAN setstats (Management Agent) utility 393
RESTMAN undeploy utility 394
RESTMAN unpublish (Application) utility 394
RESTMAN utility
deploy 381
disable 381
enable 382
getdefaults 383
getprops (Application) 383
getprops (Management Agent) 384
getstats (Application) 384
getstats (Management Agent) 386
list 386
query (Application) 387
query (Management Agent) 387
republish (Application) 389
resetdefaults (Management Agent) 388
resetprops 388
resetstats (Application) 389
resetstats (Management Agent) 391
setdefault (Management Agent) 392
setprops (Application) 393
setprops (Management Agent) 393
undeploy 394
unpublish (Application) 394
Run-time properties
for Sonic ESB services 247

S
SAML
Choosing security configuration model 336
Creating and configuring service provider metadata 337
Registering service provider 338
Security configuration model 339
Using in security configurations 336
Sample WSA Web application
moving 90
Schema holder 200
Script file
wspd.cgi.sh shell 176
Secure Socket Layer (SSL)
WSA communications 91, 287
Secure Sockets Layer 216
Secure Sockets Layer (SSL)
AppServer 52, 150
and AIA 76
Secure Sockets Layer (SSL) (continued)
and OpenEdge Adapter for Sonic ESB 256
Security
Secure Sockets Layer 52, 150
WSA 91
WSA configurations 117
WSA features 119
WSA role 118
Server authentication 76
Server Control Parameters
AppServer setting 52, 150
Server Executable File
AppServer setting 52
WebSpeed setting 150
Server Parameters
AppServer setting 52, 150
Server Pool Parameters
AppServer setting 52
choosing the settings 62
WebSpeed setting 150
Server Port Range
AppServer setting 52, 150
ServerConnect
configuring 149, 171, 232
Configuring 232
Messaging 52, 150
Service Name (-S) startup parameter
database connections 198
remote database connections 199
serviceAvailable service property 353
serviceFaultLevel service property 353
serviceLogEntryTypes service property 353
serviceLoggingLevel service property 354
Services, Sonic ESB
deploying 255
deployment Information 247
instance, creating 248
properties, editing 246, 253
run-time properties 247
OpenEdge Adapter for Sonic ESB; Web services 257
SOAP Action URI 247
SOAP Fault Processing 248
Web Service Namespace 247
sessionMode service property 354
setdefaults function
WSAMAN utility 392, 409
setprops (Service) function
WSAMAN utility 410
setprops (WSA) function
WSAMAN utility 410
Setting CLASSPATH 239
Shared memory
database connection 239
Single-user database 197
SmartDataObject support 58
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP Action URI for Sonic ESB service</td>
<td>247</td>
</tr>
<tr>
<td>SOAP Fault Processing for Sonic ESB services</td>
<td>248</td>
</tr>
<tr>
<td>Sonic ESB deploying services</td>
<td>255</td>
</tr>
<tr>
<td>Directory Service</td>
<td>246</td>
</tr>
<tr>
<td>OpenEdge Adapter for Sonic ESB</td>
<td>246</td>
</tr>
<tr>
<td>Sonic Management Console configuration</td>
<td>243</td>
</tr>
<tr>
<td>SonicMQ Discardable messages endpoints</td>
<td>238</td>
</tr>
<tr>
<td>HTTP/SOAP acceptor, adding</td>
<td>253</td>
</tr>
<tr>
<td>Load balancing</td>
<td>238</td>
</tr>
<tr>
<td>Message reuse</td>
<td>238</td>
</tr>
<tr>
<td>OpenEdge Adapter for SonicMQ; OpenEdge Adapter for Sonic ESB</td>
<td>246</td>
</tr>
<tr>
<td>Sports2000 database</td>
<td>197</td>
</tr>
<tr>
<td>SSL BrokerConnect</td>
<td>76</td>
</tr>
<tr>
<td>SSL advanced feature</td>
<td></td>
</tr>
<tr>
<td>AppServer setting</td>
<td>52</td>
</tr>
<tr>
<td>WebSpeed setting</td>
<td>150</td>
</tr>
<tr>
<td>SSL general features</td>
<td></td>
</tr>
<tr>
<td>AppServer setting</td>
<td>52</td>
</tr>
<tr>
<td>WebSpeed setting</td>
<td>150</td>
</tr>
<tr>
<td>SSL parameters</td>
<td></td>
</tr>
<tr>
<td>OpenEdge Adapter for SonicMQ setting</td>
<td>233</td>
</tr>
<tr>
<td>staleO4GLObjectTimeout service property</td>
<td>354</td>
</tr>
<tr>
<td>Starting an AppServer</td>
<td></td>
</tr>
<tr>
<td>NSMAN command-line utility</td>
<td>60</td>
</tr>
<tr>
<td>OpenEdge Management or OpenEdge Explorer</td>
<td>59</td>
</tr>
<tr>
<td>Starting configurations WSA</td>
<td></td>
</tr>
<tr>
<td>Starting a WSA instance</td>
<td>102</td>
</tr>
<tr>
<td>STARTING status</td>
<td>61</td>
</tr>
<tr>
<td>State-aware operating mode overview</td>
<td>44</td>
</tr>
<tr>
<td>State-free operating mode overview</td>
<td>44</td>
</tr>
<tr>
<td>State-reset operating mode overview</td>
<td>44</td>
</tr>
<tr>
<td>Stateless operating mode overview</td>
<td>44</td>
</tr>
<tr>
<td>Static files</td>
<td></td>
</tr>
<tr>
<td>HTML</td>
<td></td>
</tr>
<tr>
<td>UNIX 178</td>
<td></td>
</tr>
<tr>
<td>Windows 157</td>
<td></td>
</tr>
<tr>
<td>Java class files</td>
<td></td>
</tr>
<tr>
<td>UNIX 178</td>
<td></td>
</tr>
<tr>
<td>Windows 157</td>
<td></td>
</tr>
<tr>
<td>Status, AppServer run-time</td>
<td>61, 71</td>
</tr>
<tr>
<td>Status.p procedure validating configurations</td>
<td></td>
</tr>
<tr>
<td>UNIX 187</td>
<td></td>
</tr>
<tr>
<td>Stream Code Page (-cpstream) startup parameter</td>
<td>204</td>
</tr>
<tr>
<td>Syntax</td>
<td></td>
</tr>
<tr>
<td>ASBMAN utility</td>
<td>60</td>
</tr>
<tr>
<td>ASCONFIG utility</td>
<td>56, 105</td>
</tr>
<tr>
<td>System requirements</td>
<td></td>
</tr>
<tr>
<td>NT and Windows</td>
<td>51</td>
</tr>
<tr>
<td>UNIX 51, 80</td>
<td></td>
</tr>
<tr>
<td>TCP ports</td>
<td>217</td>
</tr>
<tr>
<td>Testing configuration</td>
<td></td>
</tr>
<tr>
<td>for AIA</td>
<td>77, 83</td>
</tr>
<tr>
<td>Testing utilities</td>
<td></td>
</tr>
<tr>
<td>for AIA 83, 426–428, 430</td>
<td></td>
</tr>
<tr>
<td>Tuning</td>
<td></td>
</tr>
<tr>
<td>OpenEdge Web services</td>
<td>114, 280</td>
</tr>
<tr>
<td>ubroker.properties file</td>
<td></td>
</tr>
<tr>
<td>BrokerConnect properties editing</td>
<td>234</td>
</tr>
<tr>
<td>UNIX 179</td>
<td></td>
</tr>
<tr>
<td>hierarchy</td>
<td></td>
</tr>
<tr>
<td>AppServer 56</td>
<td></td>
</tr>
<tr>
<td>UNIX 179</td>
<td></td>
</tr>
<tr>
<td>location</td>
<td></td>
</tr>
<tr>
<td>UNIX 179</td>
<td></td>
</tr>
<tr>
<td>modifying for BrokerConnect</td>
<td>234</td>
</tr>
<tr>
<td>overview</td>
<td></td>
</tr>
<tr>
<td>UNIX 179</td>
<td></td>
</tr>
<tr>
<td>Windows 157</td>
<td></td>
</tr>
<tr>
<td>structure</td>
<td>179</td>
</tr>
<tr>
<td>unique values needed</td>
<td>179</td>
</tr>
<tr>
<td>updating for a secure WSA</td>
<td>91, 287</td>
</tr>
<tr>
<td>validating edits</td>
<td>179</td>
</tr>
<tr>
<td>undeploy function</td>
<td></td>
</tr>
<tr>
<td>WSAMAN utility</td>
<td>411</td>
</tr>
<tr>
<td>UNIX requirements</td>
<td>51, 80</td>
</tr>
<tr>
<td>update function</td>
<td></td>
</tr>
<tr>
<td>WSAMAN utility</td>
<td>412</td>
</tr>
<tr>
<td>URL</td>
<td></td>
</tr>
<tr>
<td>international Web site</td>
<td>205</td>
</tr>
<tr>
<td>URLs</td>
<td></td>
</tr>
<tr>
<td>WSA 96</td>
<td></td>
</tr>
<tr>
<td>Using ClientASK property</td>
<td>68</td>
</tr>
<tr>
<td>using OpenEdge Management or OpenEdge Explorer</td>
<td></td>
</tr>
<tr>
<td>configuring</td>
<td></td>
</tr>
<tr>
<td>WebSpeed 150</td>
<td></td>
</tr>
<tr>
<td>uuid</td>
<td></td>
</tr>
<tr>
<td>in ubroker.properties</td>
<td>179</td>
</tr>
</tbody>
</table>
Validating
WebSpeed configurations
UNIX 185
Windows 165
Versioning
OpenEdge Web services 110
Video files (.mpeg, .mpg)
stored on Web server
UNIX 178
Windows 157
Viewing AIA status information
using a Java container 83, 426–428, 430
Virtual paths
for Java container's 77
using with Java container 77
using with uri 77

waitIfBusy service property 355
Web application definition
WSA 98, 271
Web server
configurations with WSA 95, 271
NSAPI type
UNIX 173
Windows 155
setting up
UNIX 173
Windows 154
Web servers
and digital certificates 79
server authentication 76
Web Service Definition (WSD) file
and OpenEdge Adapter for Sonic ESB 246
Web Service Definition file 246
Web Service Description Language file
deployment to WSA 110
providing to clients 113
Web Service Invocation
OpenEdge Adapter for Sonic ESB 241
Web Service Namespace
for Sonic ESB service 247
Web services
large messages in Java container 91
OpenEdge Web services 107
and OpenEdge Adapter for Sonic ESB 253
Web Services Adapter
WSACONFIG utility 395
WSAMAN utility
deploy 396
disable 398
enable 398
export 399
Web Services Adapter (continued)
WSAMAN utility (continued)
getdefaults 400
getprops (Service) 401
getprops (WSA) 402
getstats (Service) 402
getstats (WSA) 403
import 403
list 405
query (Service) 405
query (WSA) 406
resetdefaults 407
resetprops 407
resetstats (Service) 408
resetstats (WSA) 408
setdefault 409
setprops (Service) 410
setprops (WSA) 410
undeploy 411
update 412
Web Services Adapter (WSA)
administration
architecture 94, 269
components 95
prerequisites 94, 269
configuration
levels of 89, 261
configuring for Web services 106, 275
creating instances
multiple 102
deploying Web services 107
environments
production 90
instance
properties
for WSA instance 103, 275
instance management 103, 274
Java container/Web server configurations 95, 271
overview 32
properties
modifying 105
recognize from Java container 90
role in security 118
security configurations
reference 120
security features 119
security overview 91
starting an instance 102, 273
statistics
WSA 106, 275
testing an instance
testing configuration
for WSA 102, 273
URL construction 96
Web application definition 98, 271
Web Services Adapter (WSA) (continued)
  Web application location 90
  Web service properties, default 108, 278
  WSAMAN utility 141
Web Services Mapping (WSM) file
  and OpenEdge Adapter for Sonic ESB 246
Web Services Mapping file 246
Web site
  international 201
  multi-lingual 201
web.xml file
  digital certificates 91, 287
WebSpeed
  agent application mode 208
  ASP Messenger 223
  configuration in Windows 148
  configuration on UNIX 170
  configuring
    using OpenEdge Management or OpenEdge Explorer 150
  Debug mode 211
  development mode 207
  firewalls 217
  Messenger Administration utility 209
  passwords 214
  port numbers 212
  production mode 207
  registration mode 218
  running in Windows 164
  running on UNIX 185
  security 207
  testing after configuration
    UNIX 185
    Windows 165
  WSCONFIG utility 413
  WTBMAN utility 415
WebSpeed administration framework
  UNIX 171
  Windows 149
WebSpeed broker
  starting
    UNIX 183
WebSpeed Messenger 33
WebSpeed Transaction Server
  forcing immediate shut down
    UNIX 188
    Windows 167
  managing
    UNIX 187
    Windows 166
  overview 33
  sample instance
    Windows 149, 171
  starting
    Windows 162
WebSpeed Transaction Server (continued)
  stopping with WTBMAN
    UNIX 188
    Windows 167
  validating configurations
    UNIX 179
Weight factor
  load balancing
    Windows 161
  setting to zero
    Windows 162
Weight factors
  arbitrary
    Windows 162
  fail-over
    Windows 162
  percentage-based
    Windows 161
Windows requirements 51
Working directory
  AppServer setting 52
  environment variable
    UNIX 182
    Windows 159
  overview 58
  WebSpeed setting 150
WRKDIR environment variable
  setting for NSAPI Web server
    UNIX 175
WSA
  WSACONFIG utility 395
  WSAMAN utility
    deploy 396
    disable 398
    enable 398
    export 399
    getdefaults 400
    getprops (Service) 401
    getprops (WSA) 402
    getstats (Service) 402
    getstats (WSA) 403
    import 403
    list 405
    query (Service) 405
    query (WSA) 406
    resetdefaults 407
    resetprops 407
    resetstats (Service) 408
    resetstats (WSA) 408
    setdefault 409
    setprops (Service) 410
    setprops (WSA) 410
    undeploy 411
    updateWSAMAN utility
      update 412
WSACONFIG utility
  Web Services Adapter (WSA)
  WSCONFIG utility 105
WSAD file
  deployment to WSA 110
WSAMAN deploy utility 396
WSAMAN disable utility 398
WSAMAN enable utility 398
WSAMAN export utility 399
WSAMAN getdefaults utility 400
WSAMAN getprops (Service) utility 401
WSAMAN getprops (WSA) utility 402
WSAMAN getstats (Service) utility 402
WSAMAN getstats (WSA) utility 403
WSAMAN import utility 403
WSAMAN list utility 405
WSAMAN query (Service) utility 405
WSAMAN query (WSA) utility 406
WSAMAN resetdefaults utility 407
WSAMAN resetprops utility 407
WSAMAN resetstats (Service) utility 408
WSAMAN resetstats (WSA) utility 408
WSAMAN setdefault utility 409
WSAMAN setprops (Service) utility 410
WSAMAN setprops (WSA) utility 410
WSAMAN undeploy utility 411
WSAMAN update utility 412
WSAMAN utility
  deploy 396
  disable 398
  enable 398
  export 399
  getdefaults 400
  getprops 402
  getprops (Service) 401
  getprops (WSA) 402
  getstats (Service) 402
  getstats (WSA) 403
  import 403
  list 405

WSAMAN utility (continued)
  overview of functions 142
  query (Service) 405
  query (WSA) 406
  resetdefaults 407
  resetprops 407
  resetstats (Service) 408
  resetstats (WSA) 408
  setdefault 409
  setprops (Service) 410
  setprops (WSA) 410
  syntax 141
  undeploy 411
  update 412
WSASP 223
WSASP Messenger
  executable
    Windows 154
WSCONFIG utility
  described
    UNIX 172
    Windows 153
WSMAdmin 209
WSMAdmin page
  URL
    UNIX 188
    Windows 168
  validating configurations
    UNIX 170
    Windows 148
  viewing
    UNIX 186
    Windows 166
WTBMAN help 168, 188
WTBMAN utility
  described
    UNIX 172
    Windows 153
  note 359–361, 370–372, 374, 377, 381–382, 389,
  394, 396, 398–399, 401–403, 407, 410–412, 415