Progress®



Progress® Application Server for OpenEdge®:
Administration Guide

OpenEdge®

Copyright

© **2019 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.**

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

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Deliver More Than Expected, Icenium, Kendo UI, Kinvey, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), Sitefinity, SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. Analytics360, AppServer, BusinessEdge, DataDirect Autonomous REST Connector, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustAssembly, JustDecompile, JustMock, Kinvey Chat, NativeChat, NativeScript Sidekick, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

February 2019

Last updated with new content: Release 12.0

Updated: 2019/03/01

Table of Contents

Preface....11

Purpose....11
Audience....11
Organization....12
Using ABL documentation....12
References to ABL compiler and run-time features....13
References to ABL data types....13
Typographical conventions....13
Examples of syntax descriptions....14
Long syntax descriptions split across lines....16
Complex syntax descriptions with both required and optional elements....16
OpenEdge messages....17
Obtaining more information about OpenEdge messages....17

Chapter 1: Overview of PAS for OpenEdge administration....19

Development servers compared to production servers....20
Managing the ROOT Web application....20
Transport URLs....20
Deploy a new WebHandler Webspeed application to a PAS for OpenEdge instance....21
Tomcat logging....23
Configuration and properties files....24
Progress Application Server properties....25
PAS for OpenEdge directories....33

Chapter 2: Administrative tools....35

Tomcat utilities....36
TCMAN....36
PASOESTART....37
PASMAN....43
SECPROP....45
OEPROP....50
PASPROPCONV....53
Utilities for managing keys and digital certificates....55
OpenEdge Management and OpenEdge Explorer....56
JConsole and JMX....56
Running JConsole locally....57
Running JConsole remotely....58
Use OEJMX to manage and monitor an instance....60

configutil....72
Configutil Operations....72
Configutil.properties 75
OVERRIDE_PATHS and OVERRIDE_HOSTNAME 75
FORCE_OVERRIDE....76
DONT_OVERRIDE....76
Exporting and importing registry entries based settings....78
Exporting and Importing XML files-based settings....78
Logging of import and export operations....80

Chapter 3: Working with Instances....81
Overview of instances in PAS for OpenEdge 82
Instance management with TCMAN....83
Creating instances with TCMAN....84
Installing and running an instance as a UNIX daemon....86
Installing and running an instance as a Windows service....86
Automate lifecycle event scripts 87

Chapter 4: Application deployment and management....89
Web application management with TCMAN....89
OpenEdge Web application deployment....90
REST service deployment (deployREST.sh)....90
SOAP service deployment (deploySOAP.sh)....91
Packaging web applications....91
OpenEdge ABL applications....92
Update PROPATH in a production instance with zero downtime....92
Passing ABL error objects to REST clients....93

Chapter 5: HTTP sessions....97
Overview of HTTP sessions....97
Clusters and sticky sessions....99

Chapter 6: Load balancing options for PAS for OpenEdge....101
Apache proxy host load balancing....101
Requirements for Apache proxy load balancing....102
Configuring the Apache server for proxy load balancing....103
Configuring PAS for OpenEdge instances for proxy load balancing....104
Tomcat load balancing....105
Requirements for Tomcat load balancing....106
Configuring an Apache web server for Tomcat load balancing....107
Configuring PAS for OpenEdge for Tomcat load balancing....107
Amazon load balancing....109

Configuring Elastic Load Balancing and Auto Scaling....109

Chapter 7: Server security....111

Production instance security customizations....112
HTTPS support....112
Use SSL/TLS in PAS for OpenEdge....113
Obtain a certificate from a Certificate Authority....114
Configure a PAS for OpenEdge instance for SSL/TLS....117
Configure clients for SSL/TLS....118
The Spring Security Framework in PAS for OpenEdge....119
JVM security manager....119
Manage users and remote access....119
Role-based user authentication....120
Remote access filters....121

Chapter 8: Web application security configurations....123

Spring Security configuration files....124
Security properties files....124
URL access control file....125
Enabling security for REST, WEB, APSV and SOAP....126
User account configurations....126
Local and Extended Local user accounts....126
LDAP user accounts....128
Active Directory user accounts....128
OERealm user account configurations....129
Multi-domain support for user accounts....134
Login models....138
CORS support....139
Use CORS....140
Single sign-on (SSO) support....140
Authentication using OpenEdge Authentication Gateway....141
The STS AuthenticationProvider....141
Configuration and Testing....142
OpenEdge Authentication Gateway utilities....146
Authentication using OAuth2 and JWT....168
OAuth2 Concepts and Terms....168
OAuth2 Security Considerations....170
OAuth2 Tokens....170
JSON Web Tokens (JWT)....172
Support for OAuth2 and JWT in PAS for OpenEdge....175
Configuring a PASOE Web Application as an OAuth2 Resource Server....177
Debug Logging for OAuth2....184
Authenticate using SAML....184

SAML concepts and terms....185
Support for SAML in PAS for OpenEdge....187
Implementing SAML in a PAS for OpenEdge web application....189
Debug logging for SAML....191

Chapter 9: WebSpeed configuration and management....193

Tools for managing and configuring WebSpeed on PAS for OpenEdge....194
Deploy a classic Webspeed application to a PAS for OpenEdge instance....195
Secure the Web transport....196
Migrating classic WebSpeed applications....197

Chapter 10: TCMAN Reference....199

Using TCMAN....199
The tcman command....200
Extending TCMAN....202
Manager actions....204
List deployed applications (list)....205
Display OS and server information (info)....206
Deploy a Web application (deploy)....207
Undeploy a Web application (undeploy)....209
Reload a Web application (reload)....210
Display detailed server status (status)....211
Display memory leaks (leaks)....212
Start a Web application (enable)....213
Stop a Web application (disable)....214
Display global server resources (resources)....215
Display Web application HTTP sessions (sessions)....216
Server actions....216
Create an instance (create)....217
Delete an instance (delete)....219
Display and manage an instance's configuration (config)....220
Display or modify the server features of an instance (feature)....222
Clean up or archive server log files (clean)....224
Display server instances (instances)....224
List process ids (plist)....226
Register an instance for tracking (register)....227
Register and manage an instance as a Windows service (service)....228
Show Windows process information (showproc)....230
Start an instance (start)....231
Stop an instance (stop)....233
Display server, OS, and runtime version information (version)....234
Test a server configuration (test)....235
Stop tracking an instance (unregister)....236

Create a Tomcat worker configuration file (workers)....237
General actions....239
Display help (help)....240
Display runtime environment information (env)....240

Chapter 11: REST API Reference for oemanager.war....243

Session management....244
Get session information....244
Get client information....245
Get current request information....247
Get configuration properties....248
Update configuration properties....249
Get runtime metrics....250
Reset runtime metrics....251
Cancel a request....252
Dump a session stack....253
Terminate a session....254
Terminate idle ABL sessions....255
Agent management....256
List agents....256
Check ABL object tracking status....257
Enable or disable ABL object tracking....258
Get a report of tracked ABL objects....259
Get agent status information....260
Get session metrics....261
Get connection information....262
Get request information....264
Get configuration properties....267
Update configuration properties....268
Stop a multi-session agent....270
Add a multi-session agent....271
Transport management....272
Get transport configuration properties (SOAP/REST)....272
Update properties of a transport (SOAP/REST)....274
Enable or disable a transport (SOAP/REST)....275
Enable or disable a transport (APSV)....276
Get runtime metrics of a transport (SOAP/REST)....277
Get runtime metrics of a transport (APSV)....279
Reset runtime metrics for a transport (SOAP/REST)....280
Reset runtime metrics of a transport (APSV)....281
List deployed services (SOAP/REST)....282
Get information about a service (SOAP/REST)....283
Get configuration properties of a service (SOAP)....285
Deploy a new service (REST)....286

Deploy a new service (SOAP)....287
Update runtime properties of a service (SOAP)....288
Get runtime metrics for a service (SOAP)....289
Reset runtime metrics for a service (SOAP)....290
Enable or disable a service (SOAP)....291
Remove a service (SOAP/REST)....292
Application management....293
List ABL application information....294
Get information about a specific ABL application....296
List Web applications deployed to an ABL application....298
Get information about a specific Web application....299
Monitoring memory use....301
Track memory use....301
Tracking memory use....302
Generate memory use report for an agent....303
Generate memory use report for an ABL session....305

Index....307

Preface

For details, see the following topics:

- Purpose
- Audience
- Organization
- Using ABL documentation
- Typographical conventions
- Examples of syntax descriptions
- OpenEdge messages

Purpose

This is a guide to administering Progress® Application Server for OpenEdge® that focuses on the use of command line utilities (TCMAN) and management APIs (oemanager). For information on administration with OpenEdge Management or OpenEdge Explorer, see *OpenEdge Management: Progress Application Server for OpenEdge Configuration*.

Audience

This manual is intended for any OpenEdge application server administrators or ABL developers who need to create. configure, or administer instances of the Progress Application Server for OpenEdge.

Organization

Overview of PAS for OpenEdge administration on page 19

Introduces the administrative tools and concepts for Progress Application Server (PAS) for OpenEdge.

Working with Instances on page 81

Explains how to create and manage instances of the core PAS for OpenEdge server.

Application deployment and management on page 89

Explains how to deploy, manage, and package PAS for OpenEdge Web applications.

HTTP sessions on page 97

An overview of HTTP sessions which are a standard method for maintaining context during a series of requests/responses between an HTTP client and a Web application.

Load balancing options for PAS for OpenEdge on page 101

Describes three alternatives for configuring load balancing among PAS for OpenEdge instances.

Server security on page 111

Provides an overview of security features and concepts in PAS for OpenEdge.

Web application security configurations on page 123

Explains how to implement Spring Security models, authentication and authorization protocols, and single sign-on for Web applications deployed to a PAS for OpenEdge instance.

WebSpeed configuration and management on page 193

Describes configuration of PAS for OpenEdge for deployment of WebSpeed applications.

TCMAN Reference on page 199

A command reference for the TCMAN command line management utility.

REST API Reference for oemanager.war on page 243

An API reference describing the REST services that can be used to manage a PAS for OpenEdge instance.

Note: For additional PAS for OpenEdge documentation, including white papers on advanced topics, see:

https://community.progress.com/community_groups/openedge_development/m/documents

---

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 go to https://docs.progress.com.

References to ABL compiler and run-time features

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

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

References to ABL data types

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

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

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

Typographical conventions

This documentation uses the following typographical and syntax conventions:

Convention	Description
Bold	Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.
Italic	Italic typeface indicates the title of a document, or signifies new terms.
SMALL, BOLD CAPITAL LETTERS	Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, **GET** and **CTRL**.

Convention	Description
KEY1+KEY2	A plus sign between key names indicates a **simultaneous** key sequence: you press and hold down the first key while pressing the second key. For example, **CTRL+X**.
KEY1 KEY2	A space between key names indicates a **sequential** key sequence: you press and release the first key, then press another key. For example, **ESCAPE H**.
Syntax:	
`Fixed width`	A fixed-width font is used in syntax, code examples, system output, and file names.
`Fixed-width italics`	Fixed-width italics indicate variables in syntax.
`Fixed-width bold`	Fixed-width bold italic indicates variables in syntax with special emphasis.
`UPPERCASE fixed width`	ABL keywords in syntax and code examples are almost always shown in upper case. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class.
Period (.) or colon (:)	All statements except `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` end with a period. `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` statements can end with either a period or a colon.
[ ]	Large brackets indicate the items within them are optional.
[ ]	Small brackets are part of ABL.
{ }	Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.
{ }	Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.
|	A vertical bar indicates a choice.
. . .	Ellipses indicate repetition: you can choose one or more of the preceding items.

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

```
INITIAL [ constant [ , constant ] ]
```

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

Syntax

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

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

Syntax

```
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

```
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

```
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 }:

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

```
ASSIGN   { [ FRAME frame ] { field [ = expression ] }
           [ WHEN expression ] } . . .
       | { record [ EXCEPT field . . . ] }
```

OpenEdge messages

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

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.
- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.
- **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

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

- Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.
- Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

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

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

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

Obtaining more information about OpenEdge messages

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

- Choose **Help** > **Recent Messages** to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.
- Choose **Help** > **Messages** and then type the message number to display a description of a specific OpenEdge message.
- In the Procedure Editor, press the **HELP** key or **F1**.

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

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

1. Start the Procedure Editor:

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

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

1

Overview of PAS for OpenEdge administration

This section is an introduction to the overall administrative framework of PAS for OpenEdge.

Note: For additional PAS for OpenEdge documentation, including white papers on advanced topics, see: https://community.progress.com/community_groups/openedge_development/m/documents

For details, see the following topics:

- Development servers compared to production servers
- Managing the ROOT Web application
- Transport URLs
- Tomcat logging
- Configuration and properties files
- Progress Application Server properties
- PAS for OpenEdge directories

Development servers compared to production servers

Progress Application Server for OpenEdge can be installed as either a server for developing and testing Web applications or as a production server for application deployment.

The difference between a development server and a production server is largely a matter of security. Typically, a server in a development environment allows unrestricted access to and control by a user or group of users. A production server, on the other hand, is configured to restrict access to authorized users and to limit control to system administrators. For example, in a development environment anyone might be allowed to shut down the server, whereas, in a production environment, only an administrator with appropriate privileges would be allowed to stop a running server.

In addition, you should be aware that the development server includes a number of Tomcat and Progress configuration and management tools. A production server, to avoid unauthorized tampering, would not include those tools.

See also

Production instance security customizations on page 112

Managing the ROOT Web application

Tomcat requires a default Web application in `/webapps/ROOT`. In PAS for OpenEdge, the default root application is a deployment of `oeabl.war`. To suit the requirements of your particular development or deployment environment, you can replace `ROOT` with your own Web application and redeploy `oeabl.war`. For example, you can:

1. Delete the existing root application in a PAS for OpenEdge instance's `/webapps` folder.
2. Rename your Web application as `ROOT.war`.
3. Deploy your `ROOT.war` to the PAS instance.
4. Redeploy the default `oeabl.war` application, which can be found in `$CATALINA_HOME/extras`, under a folder other than `ROOT`.

Transport URLs

In PAS for OpenEdge, transports handle HTTP requests and are built into the default ROOT (`oeabl.war`) application. (A PAS for OpenEdge instance contains one or more OEABL Web applications.) There are four transports, APSV, REST, SOAP, and WEB. These transports handle requests from clients that use any one of those protocols for communication with a Web server.

The four transports and their assigned URLs shown in the following table.

Table 1: Transport URLs

Transport	Description	URL Path
APSV	Handles requests from clients that are using the AppServer over HTTP protocol.	`/apsv`
REST	Handles REST RPC over CFX and Came.l	`/rest`
SOAP	Handles SOAP 1.1 requests.	`/soap`
WEB	Handles requests that use standard HTTP verbs, which includes classic WebSpeed and OpenHTTP.	`/web`

Note: If you are referencing an application other than the ROOT application, the URL path is preceded by the application name (*app_name*`/web`, for example).

See also

Deploy a new WebHandler Webspeed application to a PAS for OpenEdge instance on page 21

Deploy a new WebHandler Webspeed application to a PAS for OpenEdge instance

The WEB transport handles requests and responses that use standard HTTP verbs. This includes interaction with clients such as WebSpeed and OpenHTTP.

If you want to change the default `/web` URL, you can add additional handlers and map them to different URLs, or you can change the mappings of the default handlers.

The WEB transport, which supports streaming HTTP applications, supports two default URL handlers:

- The `OpenEdge.Web.CompatibilityHandler` provides compatibility with WebSpeed SpeedScript and CGI Wrapper applications. This is the default handler used in an instance in a development environment.
- The `OpenEdge.Web.DefaultWebHandler` handler returns `405 Method Not Allowed` errors for requests that are not mapped to the handler. This is the default handler used in an instance in a production environment

You can add additional handlers in the instance's `openedge.properties` file and map them to specific URLs. For example:

```
defaultHandler=OpenEdge.Web.CompatibilityHandler
handler1=SportsHandler:/mycustomer
handler2=Sportshandler:/mycustomer/{custid}
```

In addition, you can change the URL mappings of the default handlers by editing the configuration files described in the following sections.

WEB-INF/web.xml

The WEB transport is defined as a servlet in the OEABL application's `WEB-INF/web.xml` file. Look for the section headed by `<servlet-name>OEWebServlet</servlet-name>`.

After the servlet definition, you will see a `<context-param>` and a `<servlet-mapping>` section, where you can change the default WEB URL. The example below shows in bold where the WEB URL is defined.

```
<!-- The context parameter and URL mapping must be consistent if
          you are looking to modify the URL space -->
    <context-param>
        <param-name>OEWebURL</param-name>
        <param-value>/web</param-value>

 </context-param>
    <servlet-mapping>
      <servlet-name>OEWebServlet</servlet-name>
      <url-pattern>/web/*</url-pattern>

    </servlet-mapping>
```

If you modify the default URL, make sure that both `parm-value` (referenced by the servlet) and `url-pattern` (referenced by the Tomcat server) have the same URL path.

WEB-INF/oeablsecurity-*.xml

If you changed the WEB URL path in the `web.xml` file, you must also change it in the security configuration file that you are using.

If you are using anonymous security (`oeablSecurity-anonymous.xml`), then you need to change modify the default intercept URL definition that is shown below.

```
<intercept-url pattern="/web/**"
                       access="hasAnyRole('ROLE_ANONYMOUS')"/>
```

If you are using one of the other security files, you must change the intercept URL shown above and the HTTP definition shown below.

```
<http pattern="/web/**"

          auto-config="true"
          use-expressions="true"
          create-session="stateless"
          disable-url-rewriting="true"
          authentication-manager-ref="oeablApplicationAuth"
          realm="OpenEdge" >
```

Note that the URL patterns specified in the security files must match the URLs specified in the `web.xml` file.

Tomcat logging

The core Progress Application Server (PAS) uses the standard logging technology employed by the Apache Tomcat Web server. For system logging, Tomcat uses an implementation of the Apache Commons Logging library based on the `java.util.logging`(JULI) framework. JULI is implemented in `$CATALINA_HOME/bin/tomcat-juli.jar`.

Note: This topic only applies to the standard JULI-based logging implemented for the core Tomcat Web server Web application logging may be based on a different framework (Log4j for example).

Log files are written to the `/logs` directory of a running instance (`$CATALINA_BASE/logs`). The default log files for PAS are:

- `catalina.log`, a log with entries that describe server activity
- `catalina.out`, a log for system output and standard error messages
- `localhost.log`, a log file for tracking Web application activity
- `localhost_access.log`, a log for tracking requests processed by the server
- `host_manager.log`, a log for Tomcat's `host-manager.war` Web application
- `manager.log`, a log for Tomcat's `manager.war` Web application

Log files, with the exception of `catalina.out`, are saved daily with the date appended to the filename (*`MM-DD_YYYY`*). `catalina.out` persists while the server is running. You can delete or archive all of the log files with the TCMAN `clean` action.

The default log files and logging levels are defined in `/conf/logging.properties`. For example, the following entry from `logging.properties` instantiates the `catalina.log` file:

```
1catalina.org.apache.juli.FileHandler.level = INFO
1catalina.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
1catalina.org.apache.juli.FileHandler.prefix = catalina.
```

Notice that the logging level is set to `INFO` which is the default for all log files in PAS. The following is a list of the JULI logging levels that apply to Tomcat.

Table 2: JULI logging levels for Tomcat

Level	Log content
SEVERE	Serious failures
WARNING	Potential problems
INFO	Informational messages
CONFIG	Static configuration messages
FINE	Trace messages
FINER	Detailed trace messages
FINEST	Highly detailed trace messages

The list is arranged in order by level of detail from the least (SEVERE) to the greatest (FINEST). Note that a level includes all the content of previous levels. For example, INFO specifies that the log should include informational messages, potential problems (WARNING), and serious failures (SEVERE).

Configuration and properties files

The following table lists the configuration files that are located in the Progress Application Server's /conf directory. Where noted, standard Tomcat configuration files were extended to support configuration of the Progress Application Server.

Note: Although these files can be edited in a text editor, it is always safer to use a utility like tcman.sh config to set, add, or remove properties.

Table 3: Progress Application Server properties files

appserver.properties	An added Progress properties file that sets custom Progress Application Server properties. It is called on startup by conf/server.xml and is also used by Progress web applications. For more information, open the conf/appserver.properties.README file.
catalina.properties	A standard Tomcat properties file that was extended to access common shared libraries and to supply external Java properties used by server.xml. For more information, open the conf/catalina.properties.README file.

`jvm.properties`	A standard properties file that sets JVM options for the Java Virtual Machine that a PAS instance runs in. **Note:** You cannot use `tcman.sh(.bat) config` to modify this file.
`openedge.properties`	An added Progress properties file that contains OpenEdge-specifc settings for debugging, logging, and other properties. For more information, open the `conf/openedge.properties.README` file.
`server.xml`	A standard Tomcat configuration file that was tailored for the Progress Application Server to use Java system properties that are defined externally configured in the `catalina.properties` and the `appserver.properites` files.

`web.xml`
 	
`context.xml`
	Standard Tomcat configuration files that contain defaults common to all the Web applications deployed in the server.

Progress Application Server properties

The following tables list the configuration properties for a Progress Application Server. Except where noted, use the TCMAN utility to set, add, or remove properties.

Note: Where indicated in the table, more information can be found in the Tomcat configuration reference at http://tomcat.apache.org/tomcat-7.0-doc/config/.

Table 4: PAS properties

`catalina.base`	A read-only string containing the path to the PAS instance's root directory. This property cannot be configured using TCMAN.
`catalina.home`	A read-only string containing the path to the PAS home server's root directory. This property cannot be configured using TCMAN.
`com.sun.management.jmxremote.access.file`	A readable/writable string containing the absolute path to where the PAS instance JMX console access control file is stored.
`com.sun.management.jmxremote.password.file`	A readable/writable string containing the absolute path to where the PAS instance JMX console password is stored.

com.sun.management.jmxremote.ssl	A readable/writable Boolean indicating whether SSL is enabled for JMX console connections. The possible values are true (use SSL) and false (no SSL).
java.io.tmpdir	A read-only string containing the path to the PAS home server's temp directory, where transitory files are created.
psc.as.ajp13.maxconnections	A readable/writable numeric value containing the maximum number of client connections for the AJP13 connector. When the maximum number is reached, new connections are queued. The minimum value is 100. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.ajp13.maxqueuesize	A readable/writable numeric value containing the maximum number of incoming requests in the AJP13 connector's queue. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.ajp13.port	A readable/writable numeric containing the TCP port number for the AJP13 protocol used by a load balancing server. The value can range from 1 to the maximum port number. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.ajp13.tomcatauth	A readable/writable Boolean indicating whether the AJP13 client's authentication feature is enabled. The possible values are true (enabled) and false (disabled). For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.alias	A read-only string containing the server's alias name, as used in TCMAN's -I option and with cluster and load balancing configurations. If the property contains home, then the server is a full PAS installation.
psc.as.autodeploy	A readable/writable Boolean indicating whether the PAS server automatically deploys web applications placed into its web application deployment directory (see property psc.as.webappdir). The possible values are true (autodeploy, less secure) and false (no autodeploy more secure). For more information, see "Host" in the Apache Tomcat documentation.
psc.as.clust.expireOnShut	A readable/writable Boolean indicating whether sessions expire on all nodes when one node is shut down. The possible values are true (all sessions expire) and false (do not expire). For more information, see "ClusterManager" in the Apache Tomcat documentation.

psc.as.clust.inactivetimeout	A readable/writable numeric value containing the number of seconds a Tomcat cluster session can be inactive before it is marked expired. The minimum value is 10 seconds. For more information, see "ClusterManager" in the Apache Tomcat documentation.
psc.as.clust.manager	A readable/writable string indicating the Tomcat cluster manager type. The possible values are `DeltaManager` and `BackupManager`. For more information, see "Cluster" in the Apache Tomcat documentation.
psc.as.clust.mcast.addr	A readable/writable string containing the Tomcat cluster multicast address, formatted as an IPv4 address. For more information, see "Channel/Membership" in the Apache Tomcat documentation.
psc.as.clust.mcast.dropafter	A readable/writable numeric value containing the amount of time a cluster member can be non-responsive before timing out. The minimum value is 10 milliseconds. For more information, see "Channel/Membership" in the Apache Tomcat documentation.
psc.as.clust.mcast.freq	A readable/writable numeric value containing the Tomcat cluster multicast heartbeat frequency. The minimum value is 10 milliseconds. For more information, see "Channel/Membership" in the Apache Tomcat documentation.
psc.as.clust.mcast.port	A readable/writable numeric value containing the Tomcat cluster multicast port number. For more information, see "Channel/Membership" in the Apache Tomcat documentation.
psc.as.clust.notifyListeners	A readable/writable Boolean indicating whether Tomcat notifies session listeners when sessions are created or expired across the node. The possible values are `true` (notify) and `false` (do not notify). For more information, see "ClusterManager" in the Apache Tomcat documentation.
psc.as.clust.recv.addr	A readable/writable string containing the Tomcat cluster multicast receiver address, formatted as an IPv4 address. For more information, see "Channel/Receiver" in the Apache Tomcat documentation.
psc.as.clust.recv.autobind	A readable/writable numeric value containing the number of times Tomcat will increment upward if the port in `psc.as.clust.recv.port` is unavailable . For more information, see "Channel/Receiver" in the Apache Tomcat documentation.

psc.as.clust.recv.maxthreads	A readable/writable numeric value containing the maximum number of threads in the Tomcat cluster message receiver (NioReceiver). The possible values range from 10 to 10,000. For more information, see "Channel/Receiver" in the Apache Tomcat documentation.
psc.as.clust.recv.nodelay	A readable/writable Boolean indicating whether the Tomcat cluster message receiver uses the TCP_NODELAY option. The possible values are `true` (option on) and `false` (option off). For more information, see "Channel/Receiver" in the Apache Tomcat documentation.
psc.as.clust.recv.port	A readable/writable numeric value containing the Tomcat cluster multicast receiver port number. For more information, see "Channel/Receiver" in the Apache Tomcat documentation.
psc.as.clust.recv.selectortimeout	A readable/writable numeric value containing the time in milliseconds for timeout of the Tomcat cluster message receiver (NioReceiver). The possible values range from 10 to 10,000. For more information, see "Channel/Receiver" in the Apache Tomcat documentation.
psc.as.clust.recv.timeout	A readable/writable numeric value containing the Tomcat cluster message receiver timeout in milliseconds. The minimum value is 10. For more information, see "Channel/Receiver" in the Apache Tomcat documentation.
psc.as.clust.sendoptions	A readable/writable bitmask for the Tomcat cluster channel send options. The possible values are synchronized_ack, 0x0004; asynchronous, 0x0008; and use ack, 0x0002. For more information, see "Cluster" in the Apache Tomcat documentation.
psc.as.commonlib	A readable/writable string containing a classpath of directories and JAR files that the PAS server uses with the web application class loader. The value is a comma separated list of absolute paths to directories or JAR files. For example: `/local/server/common/*.jar,/local/server/common/x.jar`
psc.as.executor.maxthreads	A readable/writable numeric value containing the maximum number of threads in the server's HTTP thread pool. The minimum value is 100. For more information, see "Executor" in the Apache Tomcat documentation.

psc.as.executor.minsparethreads	A readable/writable numeric value containing the minimum number of threads in the HTTP/HTTPS connector's thread pool. The range of values is from 10 to psc.as.executor.maxthreads. For more information, see "Executor" in the Apache Tomcat documentation.
psc.as.http.connectiontimeout	A readable/writable numeric value containing the maximum number of milliseconds the PAS will wait from when the client HTTP connection is made until the URI is received. The minimum value is 1000.
psc.as.http.maxconnections	A readable/writable numeric value containing the maximum number of client connections for the HTTP connector. When the maximum number is reached, new connections are queued. The minimum value is 100. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.http.maxqueuesize	A readable/writable numeric value containing the maximum number of incoming requests in the HTTP connector's queue. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.http.port	A readable/writable numeric containing the TCP port number for the HTTP protocol. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.https.clientauth	A readable/writable Boolean indicating whether the SSL (HTTPS) client's authentication feature is enabled. The possible values are true (enabled) and false (disabled). For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.https.connectiontimeout	A readable/writable numeric value containing the maximum number of milliseconds the PAS will wait from when the client HTTPS connection is made until the URI is received. The minimum value is 1000.
psc.as.https.keyalias	A readable/writable string containing the alias to the server's private key stored in the SSL's keystore. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.https.keypass	A readable/writable string containing the password to the server's private key stored in the SSL's keystore. For more information, see "Connector" in the Apache Tomcat documentation.

psc.as.https.maxconnections	A readable/writable numeric value containing the maximum number of client connections for the HTTPS connector. When the maximum number is reached, new connections are queued. The minimum value is 100. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.https.maxqueuesize	A readable/writable numeric value containing the maximum number of incoming requests in the HTTPS connector's queue. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.https.port	A readable/writable numeric value containing the TCP port number for the HTTPS protocol. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.https.protocol	A readable/writable string indicating the default HTTPS protocol. The possible values are SSLv3 and TLS. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.https.storeType	A readable/writable string indicating the type of keystore used for storing SSL private keys and certificates. The possible values are PKCS12 and JKS. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.msg.maxpipeline	A readable/writable numeric value containing the maximum number of pipelined HTTP keepAlive requests before the socket is closed. A value of -1 disables the limit. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.msg.maxpostsize	A readable/writable numeric value containing the maximum size in bytes of a connector's POST message body. A value of 0 disables the limit. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.msg.timeout	A readable/writable numeric value containing a connector's asynchronous timeout in milliseconds. The minimum value is 10. For more information, see "Connector" in the Apache Tomcat documentation.
psc.as.os	A read-only string indicating the system PAS was installed on. The possible values are unix and windows.
psc.as.parent	A read-only string containing the path to the home server. If psc.as.alias is home, then psc.as.parent is blank.

psc.as.shut.port	A readable/writable numeric containing the TCP port number for shutting down the server via a TCP network client. A value of -1 is only supported on UNIX for disabling shutdown using a network client (which is more secure). Windows requires a port number greater than or equal to 1. For more information, see "Server" in the Apache Tomcat documentation.
psc.as.shut.pwd	A readable/writable string containing the shared secret required by the PAS server before it will accept a shutdown command from a network client. The secret can be any non-blank ASCII string, and it is required in Windows. For more information, see "Server" in the Apache Tomcat documentation. **Caution:** The value of this property is stored as cleartext.
psc.as.stuckthreadthreshold	A readable/writable numeric value containing the maximum number of seconds to wait before a thread is considered stuck. The minimum value is 0 (off). For more information, see "Stuck Thread Detection Valve" in the Apache Tomcat documentation.
psc.as.type	A readable/writable string indicating the type of server. `standard` indicates a full PAS installation, and `instance` indicates a PAS instance.
psc.as.unpackwars	A readable/writable Boolean indicating whether the PAS server automatically unpacks WAR archives placed in its web application deployment directory (see property `psc.as.webappdir`). The possible values are `true` (automatic unpacking) and `false` (no automatic unpacking). For more information, see "Host" in the Apache Tomcat documentation.
psc.as.usersfile	A readable/writable string containing the path where the PAS server finds the local user account storage file. The path can be absolute or relative to the server (instance) CATALINA_BASE configuration value. For more information, see "Resources" in the Apache Tomcat documentation.
psc.as.version	A read-only string containing the PAS release version in the format *major-version.minor-version.patch-version*.

psc.as.webappdir	A readable/writable string containing the path for where the PAS server finds and loads web applications and WAR files from. If the path is not absolute, it is relative to the CATALINA_BASE path for the server (instance). For more information, see "Host" in the Apache Tomcat documentation.
psc.dns.name	A read-only string containing the server's fully qualified DNS name. This property cannot be configured using TCMAN.
psc.host.name	A read-only string containing the server's configured host name. This property cannot be configured using TCMAN.

The properties in the following table control the enabling and disabling of PAS features. All of them are settable using TCMAN, and the possible values are `on` and `off`.

Table 5: PAS features

AccessLog	Indicates whether client HTTP message and response code logging is enabled.
AJP13	Indicates whether the AJP13 protocol network port is enabled on the server for load balancing by a Apache HTTPD or IIS web server.
Cluster	Indicates whether Tomcat cluster support (which needs to be manually configured) is enabled.
CrawlerSessionManager	Indicates whether reduction of the impact of web crawler session creation (so as not to overrun the server) is enabled.
HTTP	Indicates whether the HTTP protocol network port is enabled on the server.
HTTPS	Indicates whether the HTTPS protocol network port is enabled on the server.
JAASRealm	Indicates whether the use of JAAS plug-in authentication modules for container user authentication (which requires manual configuration) is enabled.
JMXLifecycle	Indicates whether network ports used for remote JMX console are fixed. Otherwise, standard RMI connections are used.
LDAPRealm	Indicates whether the use of an LDAP/Active-Directory service for container user authentication (which requires manual configuration) is enabled.
RemoteAddrValve	Indicates whether connections are restricted to a list of specific TCP/IP addresses.

RemoteHostValve	Indicates whether connections are restricted to a list of specific DNS host names.
SecurityListener	Indicates whether security checking at server startup is enabled, which prevents the server from starting if any of the checks fail.
SingleSignOn	Indicates whether Tomcat's single-sign on feature is enabled.
StuckSessionValve	Indicates whether messages will be logged about requests that do not appear to be completing.
UserDatabase	Indicates whether the use of a flat file for container user authentication is enabled.

PAS for OpenEdge directories

The following table lists the PAS for OpenEdge directories added to the standard Tomcat directory structure.

Table 6: Progress Application Server directory structure extensions

$CATALINA_HOME/common/lib/	Contains general third-party libraries that are shared by a server, its instances, and its web applications.
$CATALINA_BASE/common/lib/	Contains general third-party libraries that are used by a single instance and its web applications. This directory is empty by default.
$CATALINA_HOME/extras/	Contains the WAR files of the default Tomcat web applications, host-manager.war, manager.war, and ROOT.war. It can also contain the WAR files that support Progress products. (For example, oeabl.war and oemanager.war.)

2

Administrative tools

There are a number of tools and utilities that you can use to configure and administer PAS for OpenEdge.

For details, see the following topics:

- Tomcat utilities
- TCMAN
- PASOESTART
- PASMAN
- SECPROP
- OEPROP
- PASPROPCONV
- Utilities for managing keys and digital certificates
- OpenEdge Management and OpenEdge Explorer
- JConsole and JMX
- configutil

Tomcat utilities

PAS for OpenEdge includes the standard Apache Tomcat administrative utilities (`startup`, `shutdown`, `configtest` and so on).

However, PAS for OpenEdge includes the TCMAN command line utility as an interface to administrative functionality provided by core Tomcat scripts and more. You will find extended administrative functionality that is easier to find and to use if you run TCMAN rather than running the Tomcat scripts directly. In addition, the TCMAN administrative functionality is tailored specifically for use with PAS for OpenEdge.

See also

TCMAN on page 36
OEPROP on page 50

TCMAN

TCMAN is a command-line utility for administering and managing the Progress application server. It extends the functionality that is provided by the core Apache Tomcat administrative scripts found in the server's `/bin` folder.

Syntax

```
{ $CATALINA_HOME | $CATALINA_BASE } /bin/tcman { .sh | .bat } action  [ general_options ]
 [ action_options ]
```

Parameters

`$CATALINA_HOME|$CATALINA_BASE`

`$CATALINA_HOME` is the path of thedirectory where the core PAS server is installed, typically *`OpenEdge-Install-Dir`*`/servers/pasoe` in PAS for OpenEdge. You always run TCMAN from `$CATALINA_HOME/bin` when you are creating new instances or worker configuration files. However, you can add the `-I` *`instance_alias_name`* option to specify an instance on which to apply the TCMAN action.

`$CATALINA_BASE` is the path of an instance's directory.

Note:

TCMAN automatically determines the value of `CATALINA_BASE` based on the directory where you start it. When you run it from the `/bin` directory of an instance, the value of `CATALINA_BASE` is the root directory of the instance.

If you run TCMAN from the the core PAS server (*`OpenEdge-Install-Dir`*`/servers/pasoe/bin`), the value of `CATALINA_BASE` and `CATALINA_HOME` are the same.

action

Specify which TCMAN action to invoke.

general_options

Specify one or more of the TCMAN common options that can apply to most actions.

The output of `tcman.sh help` *action* includes a list of general options that are applicable to a particular action.

action_options

Specify an option that applies to the selected *action*. These options are explained in the topics that describe each action.

See also

TCMAN Reference on page 199

PASOESTART

PASOESTART is an OpenEdge extension to the TCMAN utility that encapsulates a number of separate startup actions into a single command.

Starting a PAS for OpenEdge instance usually involves manually performing the following individual actions:

1. Executing `tcman clean -A` to clean and archive log files
2. Running `tcman start` to launch the instance
3. Executing `tcman env` to check if the instance is running
4. Polling the core server to report that the instance's startup process has completed
5. Scanning the core server and web application context logs for severe conditions that render the server unusable
6. Verifying that at least of one child OS process, in the form of an ABL multi-session Agent, is running
7. Scanning the multi-session Agent's log file for startup errors that render the Agent unusable

The PASOESTART command produces a more stable and predictable startup process by performing the TCMAN actions described in the first three steps, and then by performing the error checking and process validation described in the final four steps. It executes until the instance starts without any fatal errors, or until a timeout limit has been reached and the startup process is aborted.

The core Apache Tomcat server in PAS for OpenEdge was designed to start even when severe network connection failures or invalid web application contexts render the server unusable by clients. Using PASOESTART helps to ensure that the instance is stable and usable after startup.

Output from the PASOESTART action includes all the status and error messages generated during the various steps in the startup process.

Note: You *cannot* use PASOESTART to start an instance configured as a Windows service.

Syntax

Executing PASOESTART from PASMAN:

```
oe-install-dir/bin/pasman{ .sh | .bat } pasoestart -I instance-name
[-timeout seconds] [-v] [-silent] [-json] [-restart] [-archive] [-nostop]
```

Executing PASOESTART from TCMAN:

```
instance-dir/bin/tcman{ .sh | .bat } pasoestart
[-timeout seconds] [-v] [-silent] [-json] [-restart] [-archive] [-nostop]
```

Parameters

`-I instance-name`

The name of the instance to start.

Note: This parameter is required when you execute PASOESTART with PASMAN. This parameter is not necessary when you execute PASOESTART with TCMAN from the instance's *instance-name*`/bin` directory.

`-timeout seconds`

The number of seconds to wait for the PAS for OpenEdge instance to start. If the instance has not started fully within the time-out interval, it is considered hung and will be stopped.

If not specified, the time-out interval is set to a default of 30 seconds on UNIX servers and 120 seconds on Windows servers. If the interval is set to 0 (zero), only the current state of the specified instance is displayed. Negative values are not allowed and will generate a command-line option error.

`-v`

Display verbose execution information to the console.

.

`-silent`

Direct execution information to a log file (*instance-name*`/logs/pasoestart.log`). Note that status information is still sent to the console when you use `-silent`. See Status Information.

Note: Status information is still sent to the console when you use `-silent`. See Status Information.

`-json`

Format completion status information as a JSON object and send it to the console. The default format is multi-line TEXT output to the console.

The following shows the format of the JSON output:

```
{
"start-action":"startup-action",
"initial-state":"pasoe-state",
"initial-processes":"os-process-list",
"exit-state":"pasoe-state",
"exit-description":"execution-status",
"exit-processes":"os-process-list",
"exit-status":"exit-status",
"exit-errors":
 [ "line-of-error-text"
             .
             .
             .
   "line-of-error-text" ]
}
```

`-restart`

If the instance is already running or is in a hung state, attempt to stop it, and then execute a full startup. If the instance is already in the stopped state, this option has no effect

`-archive`

Archive existing log files. This option executes the `tcman clean -A` command.

`-nostop`

Turns off the default `stop` action, which automatically occurs after a timeout, or when PASOESTART detects errors in an instance's log files.

Note: `-nostop` can start an instance that is hung or in an unstable state. The primary purpose of this option is to force a problematic instance to start inorder to debug issues. You should not use it for standard startups.

Status Codes

PASOESTART is always a blocking operation. Control is not returned to the shell until operations are complete and the PAS for OpenEdge instance is in a known state. (See Status Information for more on known states.) After PASOESTART completes, the instance's startup completion status is in the OS error variable (`%errorlevel%` for Windows, `$?` for UNIX).

Status Code	Description
0	The instance started/restarted without errors and is usable in a `running` state.
1	The core (Tomcat) server process failed to start and the instance is left in the `stopped` state.
2	The instance started, but encountered errors that make some/all of it unusable, and is left in the `stopped` state for troubleshooting and problem resolution.
3	The instance was already in the `running` state, was not restarted, and was left in a `running` state that may, or may not, be usable.

4	The instance startup was attempted from a `stopped` state, but timed out before it reached a `running`state. The instance is left in the `stopped` state for troubleshooting and problem resolution.
5	The instance was hung in the `starting` state, was not restarted, and was left in the hung `starting` state.
6	The instance was hung in the `stopping`state, was not restarted, and was left in the hung `stopping` state.
10	Command-line line option error(s) prevented execution of PASOESTART.
11	PASOESTART was used to start a Windows Service and no startup process was attempted.
12	Missing OS process environment variables prevented PASOESTART from running and no startup process was attempted.
20	An unspecified internal programming error was encountered. Contact OpenEdge technical support for more information.

Status Information

PASOESTART always sends status information to the console, even when you use the `-silent` option. A full status report includes following information fields, which are available either as human-readable text or as a JSON object:

Field Name	Description
`Start action`	The type of startup process executed: { `start` | `restart` | `status` }
`Initial state`	The initial state of the PASOE instance found at execution time { `stopped` | `starting` | `stopping` | `running` }
`Initial processes`	A space separated list of the PASOE instance's OS process ids found at execution time: A single 0 value indicates no OS processes (i.e. `stopped`) A single non-zero value indicates the PASOE server is running but no multi-session Agents (i.e. `starting` or `stopping`) Multiple non-zero values indicts the PASOE server and at least one Agent is running – however errors may be present that prevent the instance for serving clients
`Exit state`	The PASOE instance’s state when the pasoestart command completes. See Initial state
`Exit description`	A text description of the command completion status

`Exit status`	A single numerical value of the pasoestart command's completion status. See the above list of **pasoestart** exit statuses
`Exit errors`	A brief list of error or fatal log file entries found in the PASOE instance's /logs files. This is provides as hints to why the PASOE instance startup failed and provides references for where the administrator/developer may find the full error output. The Exit errors information may be blank/empty for some types of completion statuses

Server State Transitions

PASOESTART may transition the state of a PAS for OpenEdge instance zero or more times during its execution. The following table shows the different initial and exit state transitions an instance may go through.

Initial State	Exit State	Description
`stopped`	`started`	PASOESTART succesfully started Tomcat, the OpenEdge application, the Agent, and startup event procedures
`stopped`	`stopped`	Startup failed and the instance was stopped due to a PASOETART timeout or errors.
`stopped`	`starting`	A startup timeout was detected and the server is hung in the starting state after a failed attempt to stop the instance. Try the PASOESTART `restart` command once, then manually stop the instance if restart is not successful.
`started`	`started`	The instance was already running – the state of the instance is not changed. The server may, or may not, be usable
`started`	`started`	After a PASOESTART `restart`, a running instance was stopped, and then Tomcat, the OpenEdge application, the Agent, and startup event procedures were restarted successfully.
`started`	`stopped`	After a PASOESTART `restart`, a running instance was stopped, and failed during a restart due to a timeout or errors. Try the PASOESTART `restart` command once more, then manually stop the instance if restart is not successful.
`started`	`starting`	After a PASOESTART `restart`, a running instance was stopped, and timed out in the`starting` state. The instance is not in a usable condition. Retry the PASOESTART `restart` command once. If it is not successful, manually stop the instance.

starting	starting	After a PASOESTART restart, an instance is hung in the startingstate and no restart was performed. The instance is not in a usable condition. Retry the PASOESTART restart command once. If it is not successful, manually stop the instance.
starting	starting	The PASOE instance is hung in the starting state and no restart was performed. The instance is not in a usable condition. Manually stop the instance or run a PASOESTART restart.
starting	stopping	After attempting to stop an instance hung in the starting state, the instance hangs in the stopping state. The instance is not in a usable condition. Try the PASOESTART restart command once. If it is not successful, manually stop the instance.
starting	stopped	After a PASOESTART restart, an instance hung in the starting state, was stopped, and failed to restart due to a timeout our errors. Retry the PASOESTART restart command once.
starting	started	After a PASOESTART restart, an instance hung in the starting state was stopped and then Tomcat, the OpenEdge application, the Agent, and startup event procedures were restarted successfully.
stopping	stopping	The instance is hung in the stopping state and no restart was performed. The instance is not in a usable condition. Manually stop the instance or run a PASOESTART restart.
stopping	stopping	After a PASOESTART restart, an instance hung in the stopping state , was stopped, restarted, and then hung in the stopping state because of a timeout or errors. The instance is not in a usable condition. Manually stop the instance.
stopping	stopped	After a PASOESTART restart, an instance hung in the stopping state , was stopped, restarted, and stopped a second time because of a timeout or errors. The instance is not in a usable condition.
stopping	started	After a PASOESTART restart, an instance hung in the stopping state, was stopped, and then successfully restarted and then Tomcat, the OpenEdge application, the Agent, and startup event procedures were restarted successfully.

PASMAN

PASMAN is a command-line utility that always invokes TCMAN from the core PAS for OpenEdge installation(*OpenEdge-Install-Dir*/servers/pasoe/bin). You can apply TCMAN actions to an instance by using the -I *instance_name* option with PASMAN.

The script for PASMAN is located in *OpenEdge-Install-Dir*/bin which is usually included in your PATH environment variable.

PASMAN takes the same options and actions as TCMAN. However, you must specify an instance name (using -I *instance_name*) whenever an action applies to a specific instance.

Syntax

```
pasman action  [general_options] [action_options ]
```

Parameters

action

Specify which TCMAN action to invoke. See the TCMAN Reference on page 199 for information on actions.

general_options

Specify one or more of the TCMAN common options that can apply to most actions.

The output of tcman.sh help *action* includes a list of the general options that are applicable to the specified *action*.

Important:

When an action affects a specific instance, you must explicity reference the instance using the `-I` *`instance_name`* general option with PASMAN. If you do not specify an instance name, the action applies to the core server in `$CATALINA_HOME`. The actions where you can use `-I` with PASMAN to specify a particular instance are:

Action	Description
clean	Clean up or archive log files.
config	Display and manage configurations
deploy	Deploy a Web application
disable	Stop a Web application
enable	Start a Web application
env	Display runtime environment information
feature	Display or modify server features
info	Display OS and server information
leaks	Display memory leaks
list	List deployed applications
plist	List process ids
reload	Reload a Web application
resources	Display global server resources
service	Register and manage an instance as a Windows service
sessions	Display Web application HTTP sessions
showproc	Show Windows process information
start	Start an instance
status	Display detailed server status
stop	Stop an instance
test	Test a server configuration
undeploy	Undeploy a Web application
version	Display server, OS, and runtime version information

Other actions only apply to the core PAS for OpenEdge installed in *OpenEdge-Install-Dir*/servers/pasoe/bin and not to instances. These core server actions, where -I *instance_name* should *not* be used with PASMAN, are:

Action	Description
create	Create an instance
delete	Delete an instance
instances	Display server instances
register	Register an instance for tracking
unregister	Stop tracking an instance
workers	Create a Tomcat worker configuration file

action_options

Specify an option that applies to the selected *action*. These options are explained in the topics that describe each action.

See also

TCMAN Reference on page 199

SECPROP

SECPROP is a command-line utility that you can use to manage properties in the oeablSecurity.properties file.

What is the oeablSecurity.properties file?

The oeablSecurity.properties file provides an easy way for you to configure security properties for a PAS for OpenEdge web application. It acts as an interface to the Spring Security framework that guards the web application. Instead of changing Spring bean constructor and property settings in multiple XML files, you can manage all security properties for your web application from one single location in the oeablSecurity.properties file.

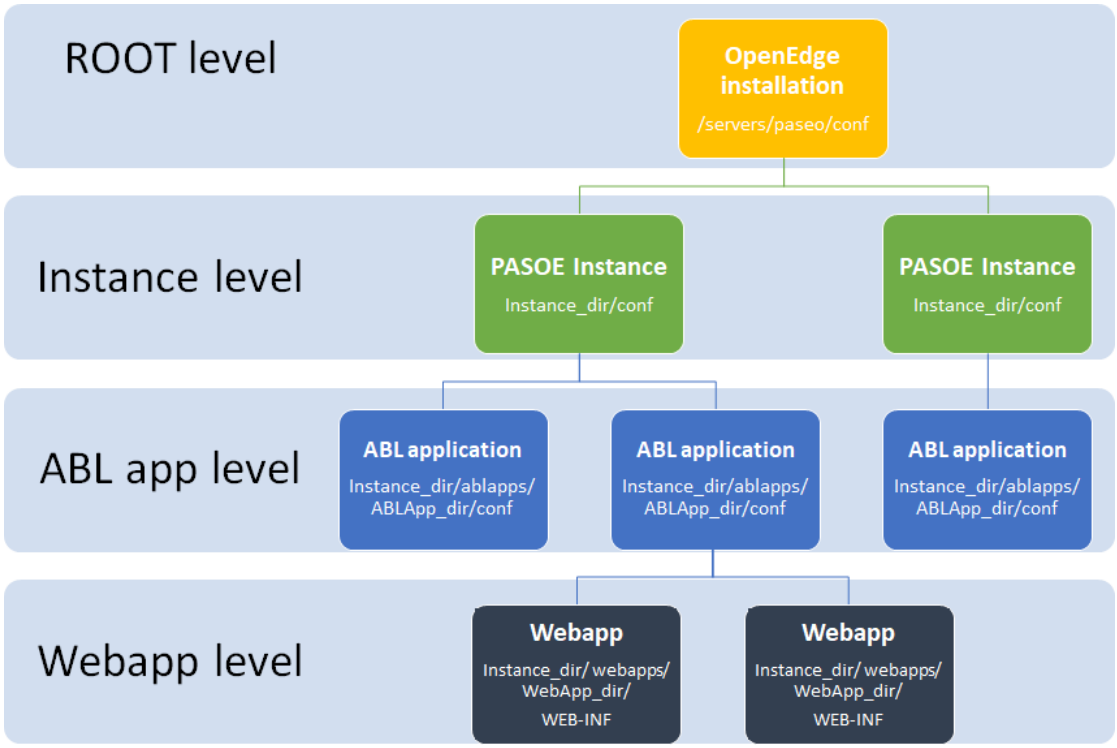
Security property levels

While the purpose of the oeablSecurity.properties file is to secure a PAS for OpenEdge web application, the security properties are distributed and replicated across four hierarchical levels:

- In *OpenEdge_Install_Directory*/servers/pasoe/conf: When you install OpenEdge, a copy of the oeablSecurity.properties file is created in this directory. This is a master file that contains ALL security properties that can be set for a web application.

- In *PASOE_instance_directory*/conf: When you create a PAS for OpenEdge instance, a copy of the oeablSecurity.properties file is created in this directory from the master file. This file inherits a subset of the master file's properties.
- In *PASOE_instance_directory*/ablapps/ABLApp_directory/conf: When you deploy an ABL application to a PAS for OpenEdge instance, a copy of the oeablSecurity.properties file is created in this directory. It inherits properties from the instance-level security properties file.
- In *PASOE_instance_directory*/webapps/WebApp_directory/WEB-INF: When you deploy a web application for an ABL application to a PAS for OpenEdge instance, a copy of the oeablSecurity.properties file is created in this directory. It inherits properties from the ABL application-level security properties file.

The inheritance architecture looks something like this:



There is a parent-child relationship between each level in this architecture. If you modify security properties at the parent level, the changes are propagated to any new children.

For example, if you modify the oeablSecurity file for a PAS for OpenEdge instance, any new ABL applications that are deployed to the instance get a copy of the updated file. Similarly, if you modify security properties at the ABL application level, any new web applications that are created for that ABL application get an updated copy of the ABL application's oeablSecurity file.

Note: You cannot modify the oeablSecurity.properties file at the top-most level, that is, in the *OpenEdge_installation*/servers/pasoe/conf directory using the SECPROP utility. Further, if you modify properties at any of the 'child' levels, any changes to the parent-level properties will apply only to new children of the same parent.

Syntax

```
secprop log_level(Optional) operation_type_parameter filter operation
```

Log level

Use the parameters listed below to optionally set the logging level. By default, log messages are written to the command line console. However, you can modify this behavior to write the log output to a file by configuring the `secproputil-logging.xml` file that you can find in the `conf` folder in a PAS for OpenEdge instance's directory, and also in the `/servers/pasoe/conf` directory in the OpenEdge root installation folder.

`-v`

Sets the logging level to INFO. This is the default level. Typically returns a single line log message for each operation performed or error thrown.

`-g`

Sets the logging level to DEBUG. Returns debug messages about the operation that was performed or attempted and the files that were accessed. Also includes INFO-level messages.

`-t`

Sets the logging level to TRACE. Returns detailed messages about each method that was called internally while attempting to perform an operation. Also includes DEBUG and INFO-level messages.

Operation type parameters

`-h/-help`

Displays the syntax and parameter descriptions for the SECPROP command.

`-m/-manage`

Specify this parameter if you want to perform list, update, add, or delete operations.

`-r`

Specify this parameter to resolve a property value across hierarchical levels. The SECPROP utility starts by attempting to find the property in the `oeablSecurity.properties` file for specified web application, ABL application, or PAS for OpenEdge instance. If it does not find it, it attempts to find it in the parent level security properties file, and then at the grandparent level and so on, until it reaches the root installation level. If the property is not found at any of the levels, an error is displayed.

Filters

`-c`

Use this filter if you want to list properties in the master `oeablSecurity.properties` file.

`-i` *`PASOE_instance_name`*

Use this filter to specify the name of a PAS for OpenEdge instance. For example, `-i oepas1`. The operation that you perform (listing, updating, adding, or deleting security properties) will apply only to this PAS for OpenEdge instance.

`-a` *`PASOE_instance_name.ABLApp_name`*

Use this filter to specify the name of an ABL application. For example, `-a oepas1.myABLApp`. The operation that you perform (listing, updating, adding, or deleting security properties) will apply only to this ABL application.

`-w` *`PASOE_instance_name.ABLApp_name.Webapp_name`*

Use this filter to specify the name of a web application. For example, `-w oepas1.myABLApp.myWebApp`. The operation that you perform (listing, updating, adding, or deleting security properties) will apply only to this web application.

You can also concatenate filter parameters to perform operations at multiple levels simultaneously.

`-ia` *`PASOE_instance_name.ABLApp_name`*

For example, `-ia oepas1.myABLApp`. This enables you to define operations (listing, updating, adding, or deleting security properties) that apply to the PAS for OpenEdge instance as well as the ABL application.

`-iaw PASOE_instance_name.ABLApp_name.Webapp_name`

For example, `-iaw oepas1.myABLApp.myWebApp`. This enables you to define operations (listing, updating, adding, or deleting security properties) that apply to the PAS for OpenEdge instance, the ABL application, as well as the web application.

Read operations

To read the value of a property, simply enter the property name. For example:

```
secprop -m -i oepas1 http.all.realm
```

To list all the properties of an `oeablSecurity.properties` file, use the `all` switch. For example:

```
secprop -m -i oepas1 -all
```

To read the value of a property at multiple hierarchical levels, combine `i`, `a`, and `w` filter parameters. For example:

```
secprop -m -iaw oepas1.myABLApp.myWebApp http.all.realm
```

You can also read the values of different properties at different levels as shown in this example:

```
secprop -m -iaw oepas1.myABLApp.myWebApp i:http.all.realm a:OEClientPrincipalFilterKey
 w:OECORSFilter.allowAll
```

To find the value of a property or to find if the property exists in an `oeablSecurity.properties` at any of the parent levels of the specified web application, ABL application, or PAS for OpenEdge instance, use the `-r`operational type parameter. For example:

```
secprop -r oepas1.myABLApp.myWebApp w:OECORSFilter.allowAll
```

You can also use the `-r` operational type parameter to find the value of different properties as shown in this example:

```
secprop -r oepas1.myABLApp.myWebApp i:http.all.realm a:OEClientPrincipalFilterKey
w:OECORSFilter.allowAll
```

Update operations

To update the value of a property, simply enter the property name and value pair. For example:

```
secprop -m -i oepas1 http.all.realm=OpenEdge
```

To update the value of a property at multiple levels, combine `i`, `a`, and `w` filter parameters. For example:

```
secprop -m -iaw oepas1.myABLApp.myWebApp http.all.realm=OpenEdge
```

You can also update the values of different properties at different levels as shown in this example:

```
secprop -m -iaw oepas1.myABLApp.myWebApp i:http.all.realm=OpenEdge
a:OEClientPrincipalFilterKey=myKey w:OECORSFilter.allowAll=false
```

Note: You can update a property only if the property already exists in the `oeablSecurity.properties` file that you are trying to modify. If the property does not exist, the update operation will fail for that file and the SECPROP utility will throw an error.

Add operations

To add a property, use the `+` switch in conjunction with a filter parameter. For example:

```
secprop -m -i oepas1 +i:http.all.realm=OpenEdge
```

To add a property at multiple levels, combine the `i`, `a`, and `w` filter parameters. For example:

```
secprop -m -iaw oepas1.myABLApp.myWebApp http.all.realm=OpenEdge
```

You can also add different properties at different levels, by using filter parameters:

```
secprop -m -iaw oepas1.myABLApp.myWebApp +i:http.all.realm=OpenEdge
+a:OEClientPrincipalFilter.domain=progress +w:OECORSFilter.allowAll=true
```

Note: You can add a property only if the property does not already exist. If the property exists in an `oeablSecurity.properties` file, the add operation will fail for that file and the SECPROP utility will throw an error.

Delete operations

To delete a property, use the ~ switch in conjunction with a filter parameter. For example:

```
secprop -m -i oepas1 ~i:http.all.realm
```

To delete a property at multiple levels, combine the `i`, `a`, and `w` parameters. For example:

```
secprop -m -iaw oepas1.myABLApp.myWebApp http.all.realm
```

You can also delete different properties at different levels:

```
secprop -m -iaw oepas1.myABLApp.myWebApp ~i:http.all.realm
~a:OEClientPrincipalFilter.domain ~w:OECORSFilter.allowAll
```

Performing multiple operations

You can also perform multiple operations in the same SECPROP command, using the relevant parameters and switches as shown in this example:

```
secprop -m -iaw oepas1.myABLApp.myWebApp ~i:http.all.realm
+a:OEClientPrincipalFilter.domain=progress w:OECORSFilter.allowAll
```

OEPROP

Purpose

OEPROP is a command-line utility used to display and manage properties in the server's `conf/openedge.properties` file.

Note: Use TCMAN (tcman config) to manage the properties in `conf/appserver.properties` or to view the properties in `conf/jvm.properties`.

Syntax

```
{ $CATALINA_HOME | $CATALINA_BASE } /bin/oeprop [ .sh | .bat ] [ -help ]
 [ group_name.prop_name | group_name.prop_name=value | +group_name.prop_name=value
 | ~group_name.prop_name | ~group_name ] [ -R ] [ -f filename ] [ -v ] [ -g ] [ -t ]
```

Parameters

`-help`

Display the syntax and parameter descriptions for the `oeprop` command.

`group_name.prop_name`

Display the value of the specified property the the specified group in `.../conf/openedge.properties`.

`group_name.prop_name=value`

Set the value of a property in a group that exists in `.../conf/openedge.properties`.

`+`*`group_name.prop_name=value`*

Add a new property to the specified group in `.../conf/openedge.properties` and set its value.

`~`*`group_name.prop_name`*

Remove the specified property in the specified group in `.../conf/openedge.properties`. If *`prop_name`* refers to a group, the group must be empty. To remove non-empty groups, use the `-R` option.

`~`*`group_name`*

Remove the specified group in `.../conf/openedge.properties`. The group must be empty. To remove non-empty groups, add the `-R` option.

`-R`

Use with `~`*`group_name`* to remove non-empty groups.

`-f` *`filename`*

Merge the contents of the text file, *`filename`*, with the existing `openedge.properties` file.

This option is particularly useful when deploying and configuring new Web applications. It allows you to add property names and property groups in one step.

`-v`

Display verbose information when `oeprop` runs.

`-g`

Display debug information when `oeprop` runs.

`-t`

Display detailed trace information when `oeprop` runs.

Note

Properties in the `openedge.properties` files are organized in groups. For example, the following shows the properties listed under the `AppServer.SessMgr` group:

```
[AppServer.SessMgr]
    agentExecFile=${psc.as.oe.dlc}/bin/_mproapsv
    agentListenerTimeout=300000
    agentLogEntryTypes=
    agentLogFile=
    agentLoggingLevel=2
    agentLogThreshold=0
    agentNumLogFiles=3
    agentStartupParam=-T ${catalina.base}/temp
    connectionWaitTimeout=3000
    idleAgentTimeout=300000
    idleConnectionTimeout=300000
    idleResourceTimeout=0
    idleSessionTimeout=300000
    ipver=IPv4
    maxABLSessionsPerAgent=200
    maxAgents=10
    maxConnectionsPerAgent=16
    numInitialAgents=1
    publishDir=${catalina.base}/openedge
    requestWaitTimeout=15000
    socketTimeout=3000
```

When you specify a property as a parameter to the `oeprop` command, the group name must precede the property name. For example: `AppServer.SessMgr.maxAgents`.

Examples

- Display the maximum number of agents (returns 10, the default setting):

```
proenv> bin\oeprop.bat AppServer.SessMgr.maxAgents
10
```

- Set the maximum number of agents to 12 and verify the change:

```
 proenv> bin\oeprop.bat AppServer.SessMgr.maxAgents=12
proenv> bin\oeprop.bat AppServer.SessMgr.maxAgents
12
```

- Add the groups and properties contained in `NewWebAppProperties.txt` to `conf/openedge.properties`:

```
 proenv> bin\oeprop.bat -f NewWebAppProperties.txt
```

PASPROPCONV

Purpose

The `paspropconv` tool is useful for migrating application logic from a Classic AppServer to a Progress Application Server for OpenEdge (PAS for OpenEdge) instance. The tool takes as input the properties of a Classic AppServer from either a `ubroker.properties` file or a `.merge` file (created using the `mergeprop` utility) and generates several files, including:

- `pasoename.ubrokername.oemerge`: This file contains two sections. The **first section** explains all the conversions made while adapting the Classic AppServer properties to PAS for OpenEdge. It also indicates properties that were not adapted, properties that you should review and modify, and other important information. Please take time to read this section. The **second section** can be used to add the new values to your PAS for OpenEdge instance using the `oeprop` utility.
- `pasoeAppName_setenv.sh` and `pasoeAppName_setenv.bat`: These environment files are generated, if there is a corresponding `[Environment.ubrokerName]` section in the `ubroker.properties` or the `.merge` file. Copy the platform-specific file to the instance's bin directory to have the environment variables read at the time your PAS for OpenEdge instance starts.
- `paspropconv.log`: Review this file for any errors that may have occurred during the conversion process.
- `paspropconv_notesdb.en`: This file is used to create the comments section of the `pasoename.ubrokername.oemerge` file. Presently, it is available only in English.

Note: The `paspropconv` utility requires `$DLC/bin/mergeprop.bat` to work. Therefore, it is bundled with the Classic AppServer in OpenEdge 11.7.3. If you are using an earlier version of the Classic AppServer, use the `mergeprop -listall` command to create a .merge file. Then, you can use the `.merge` file for conversion using `paspropconv`.

For more information, see *Progress Application Server for OpenEdge: Quick Start Guide*.

Syntax

```
paspropconv --ubrokerPropsFile absolute-path-of-ubroker.properties

--ubrokerMergeFile name-of-the-ubroker-merge-file --ubrokerName

UBroker.AS.ubrokername --pasoeAppName ABL-application-name

--pasoeWebAppName ABL-web-app-name --pasoeMergeFile pasoe-merge-filename

pasoeSetEnvFile name-of-setting-evn-variables-file

convNotesDBFile name-of-the-file-containing-conversion-notes

--logFile log-filename --loggingLevel logging-level

--logAppend 0-or-1 mergepropExe name-of-mergeprop-utility

--help
```

Parameters

`--ubrokerPropsFile`

Specify the absolute path of the `ubroker.properties` file. This parameter is required, unless you use `ubrokerMergeFile`.

`--ubrokerMergeFile`

Specify the absolute path of the `ubrokerMergeFile` that is generated using `mergeprop`. This parameter is required, unless you use `ubrokerPropsFile`.

`--ubrokerName`

Specify the fully qualified name of the AppServer broker (`UBroker.AS.brokername`).

`--pasoeAppName`

Specify the name of the ABL application to which you want to migrate your application logic. If you want to migrate the application logic to the default ABL application, specify the name of your PAS for OpenEdge instance.

`--pasoeWebAppName`

It is an optional parameter. Specify the name of the ABL web application, inside the ABL application that you already specified, to which you want to migrate your application logic. By default, the application logic will be migrated to the ROOT ABL web application.

`--pasoeMergeFile`

It is an optional parameter. Specify the name of the file that is one of the expected file outputs of running the `paspropconv` tool. You will subsequently merge the contents of this file with your PAS for OpenEdge instance's openedge.properties file. By default, the file will be named as `pasoeAppName.ubrokername.oemerge`.

`--pasoeSetEnvFile`

It is an optional parameter. Specify the name of the file that you will later copy to the PAS for OpenEdge instance's bin folder, so that environment variables are properly set when the instance is started or restarted. This file is generated on running `paspropconv`. By default, the generated files are named as `pasoeAppName_setenv.sh/.bat` (for UNIX/Windows).

`--convNotesDBFile`

It is an optional parameter. Specify the name of the file. This file is generated on running `paspropconv`. By default, the generated file will be named as `paspropconv_notesdb.en`.

`--logFile`

It is an optional parameter. Specify the name of the `paspropconv` tool's log. By default, the log is named as `paspropconv.log`.

`--loggingLevel`

It is an optional parameter. This parameter determines how much information is captured in the tool's log. Possible values are `0`, `1`, `2`, `3`, `4`, and `5`. By default, the parameter is set to `2`.

`--logAppend`

It is an optional parameter. This parameter determines whether information captured for every execution of `paspropconv` replaces the existing information in the log or appends to the end of the log. If you want information replaced, specify `0`. If you want information appended, specify `1`, which is the default value.

`--mergepropExe`

It is an optional parameter. Specify the full pathname of the `mergeprop` utility, if you are not using the default location (`<install-dir>/bin/mergeprop.bat`). The default value is `mergeprop`.

`--help`

If you need help with the `paspropconv` tool, run `paspropconv --help` to see the complete list of parameters available with the tool.

Utilities for managing keys and digital certificates

The following OpenEdge utilities, typically run from the PROENV command line, allow you to install and manage keys and digital certificates:

- `certutil` — Provides all the functions necessary to install and manage root certificates from any Certification Authority (CA) as entries in the root certificate store of an OpenEdge client machine (located in *`OpenEdge-Install-Dir`*`\certs`).

- `gendomreg` — Creates an encrypted domain registry file that can be used to seal a client-principal token. It is most often used when the token must be validated against multiple domains.
- `genpassword` — Accepts the clear-text value of a password and generates the encoded and encrypted form for the specified password.

  Note: Also see the `stspwdutil` utility in the *Getting Started: OpenEdge Authentication Server Guide*, `stspwdutil` features similar functionality but better encryption for the PAS for OpenEdge and OpenEdge Authentication Gateway servers.

- `mkhashfile` — Provides a simple way to install a root certificate in the OpenEdge root certificate store of a client machine. Such a certificate can be authorized by your own internal-use Certification Authority (CA) or by any CA that can provide you with a PEM-encoded certificate.
- `pkiutil` — Provides all of the functions necessary to create and manage key store entries for OpenEdge SSL servers. It creates these entries from pairs of private keys and digital certificates that it stores in the OpenEdge server key store (located in *`OpenEdge-Install-Dir`*`\keys`).

You can display usage information for any of these utilities by running them with the `-help` option. For more detailed information, see the *Command and Utility Reference* in *OpenEdge Getting Started: Installation and Configuration*.

.

See also

Obtain a certificate from a Certificate Authority on page 114

OpenEdge Management and OpenEdge Explorer

Much of the functionality supplied by command-line utilities like TCMAN and OEPAS is also implemented in the OpenEdge Management (OEM) and OpenEdge Explorer (OEE) browser-based tools. You can also use OEM/OEE to monitor PAS server performance and display performance statistics.

For more information, see *Configuring and Administering PAS for OpenEdge with OEE/OEM*.

JConsole and JMX

JConsole

JConsole (the Java Monitoring & Management Console), which is an application included with Java, can be used to monitor and manage PAS for OpenEdge instances. JConsole is usually used on development servers for debugging. On production servers, it is commonly used for monitoring and tuning, and for removing hung sessions.

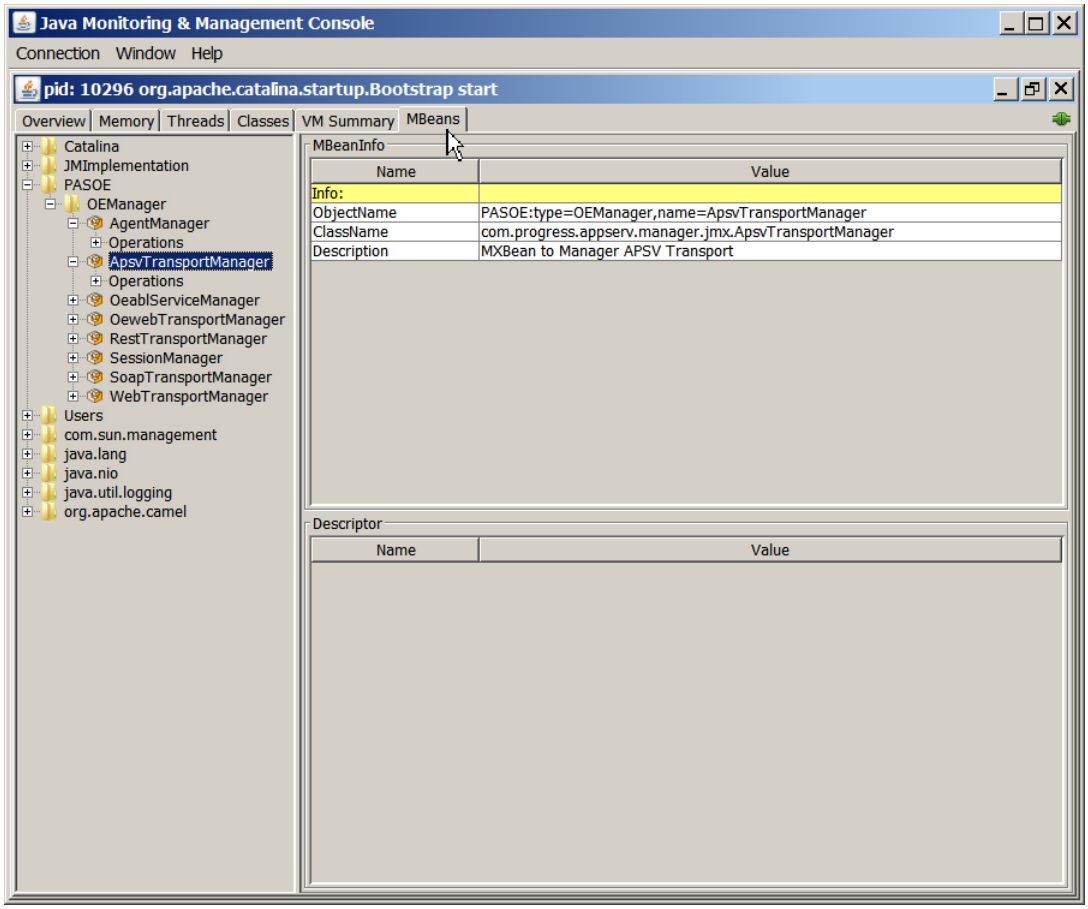
You should not use JConsole on a local production server because it uses significant resources. Using JConsole with a remote connection to a production server avoids this problem. However, enabling remote connections requires more attention to security since it opens the door to unauthorized access to the server.

For more information, see Running JConsole locally on page 57 and Running JConsole remotely on page 58.

JMX

The Progress Application Server supports the use of The Java Management Extensions (JMX) technology, which facilitates dynamic access to applications and other resources. This access is accomplished through the use of Java objects called *Managed Beans*, or *MBeans*. One or more MBeans instrument a resource through the use of the MBean's attributes, actions (defined as methods), and notifications.

The instrumented resources are listed in the **MBeans** tab of the JConsole interface. For example:



For more detailed information on JMX and MBeans, see the Java documentation at http://docs.oracle.com/javase/8/docs/technotes/guides/jmx/index.html.

OEJMX

OEJMX is a command-line utility that you can use to request information about Mbeans and to run Mbean queries. Mbeans (or Managed Beans) are the interfaces used for managing the resources of a PAS for OpenEdge server.

The output of OEJMX requests and queries is written to a file.

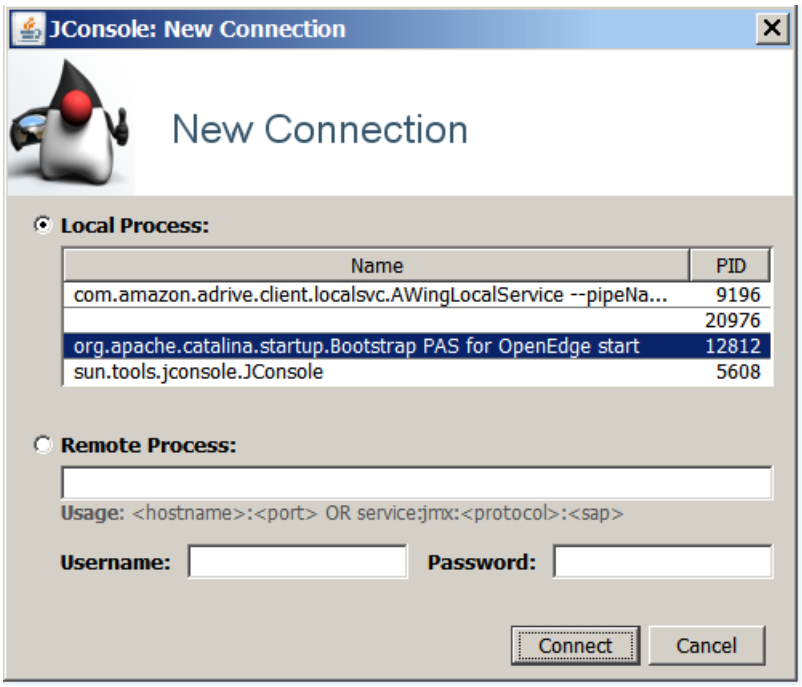
Running JConsole locally

Typically, you would run JConsole locally to monitor applications running on a PAS for OpenEdge instance in a development environment. Running JConsole locally to monitor deployed applications on a production server consumes too many resources and would adversely affect performance.

1. Start PROENV and enter `jconsole` on the command line.

The executable, `jconsole.exe` resides in the *`OpenEdge-Install-dir`*`\jdk\bin` folder, which should be in your PROPATH.

The JConsole New Connect dialog opens, which should look similar to the following:



2. Select **Local Process**.
3. Select the PAS for OpenEdge process from the list of Java processes.

   The Java Monitoring & Management Console appears.

Running JConsole remotely

One of the reasons for running JConsole remotely is to minimize the effect of JConsole on server performance, since JConsole does consume a lot of system resources. However, you should be aware of the security impact of running JConsole remotely and take appropriate measures to guard against unauthorized access.

This procedure tells you how to enable JMX remote access to a PAS for OpenEdge instance, which is a prerequisite for running the JConsole administrative tool remotely.

1. Turn on the `JMXLifecycle` feature in a PAS for OpenEdge instance's `/conf/server.xml` file.

   The best way to turn on the feature is to use the instance's TCMAN utility. For example:

   ```
   $CATALINA_BASE/bin/tcman feature JMXLifecycle=on
   ```

   TCMAN enables the JMX remote process and also opens ports for remote connection (`JNDI 10001` and `RMI 10002`). You can change these ports by manually editing the `JMXLifecycle` section in `server.xml`

2. Add the name of the instance's host to its `/conf/appserver.properties` file.

Add and set the following property in `appserver.properties`:

```
java.rmi.server.hostname=hostname
```

Note: If you do not set `java.rmi.server.hostname`, the default `localhost` will be used.

3. Restrict read/write access to the instance's `/conf/jmxremote.password` file to the user with owner privileges.

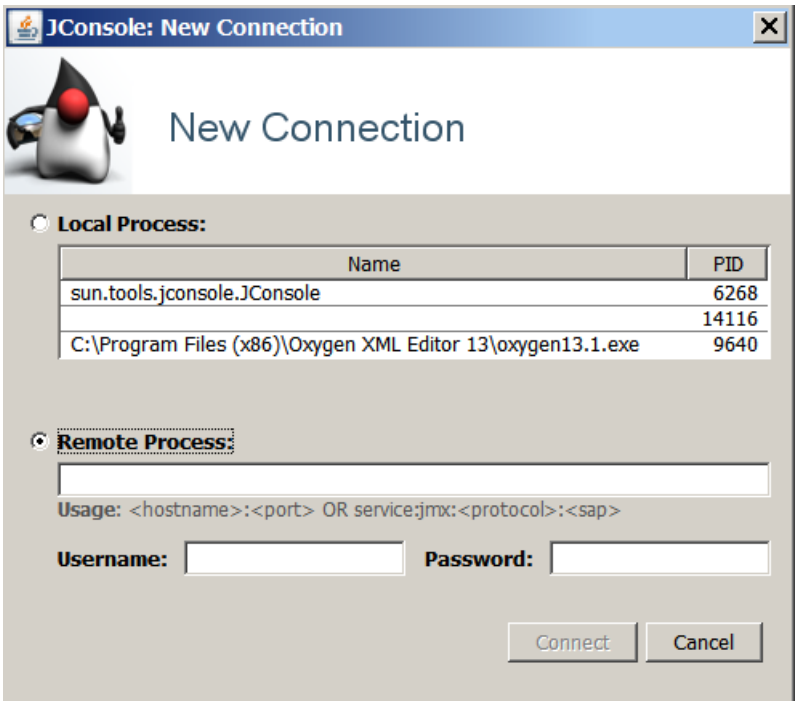
Important: `jmxremote.password` is a plain-text file that contains the username/password combination that you need in order to log in to the instance from a remote JConsole. The default username and password (`jmxadmin`, `jmxadmin`) should be changed for security reasons.

On Windows, you can change permissions on the **Security** tab of the file's **Properties** dialog. On a UNIX system, you would run the following command:

```
chmod 600 jmxremote.password
```

4. On a remote machine, browse to `$JAVA_HOME/bin` and run `jconsole.exe`.

The JConsole New Connection dialog appears.



a) Click the **Remote Process** button.

b) Specify hostname and port of the instance that you want to manage.

c) Enter the credentials that are specified in the instance's `jmxremote.password` file.

Upon successful login, the Java Monitoring & Management Console appears.

Use OEJMX to manage and monitor an instance

You can monitor and manage applications using Java Monitoring and Management Console (JConsole) and OEManager. You can also monitor and manage instances through scripts using the OpenEdge JMXcommand line utility, `OEJMX` , to connect to a local PAS for OpenEdge instance. Based on JMX (Java Management Extensions), `OEJMX` connects to an instance using Java objects called Managed Beans, MBeans or simply beans. One or more beans instrument a resource using attributes and operations. This direct access to the underlying beans provides administrators with the freedom to write scripts to monitor and manage applications in ways that may not be available within OpenEdge Management or JConsole.

See also:

- List available MBeans to locate AgentManager details
- Write a custom OEJMX query to get agents
- Get agent information
- Generate a stack trace
- Common OEJMX queries
- OEJMX utility

List available MBeans to locate AgentManager details

A running PAS for OpenEdge instance must be available.

To monitor and manage an instance using OEJMX, it is helpful to have a complete list of available MBeans for an instance. Use the OEJMX utility with the `-C` parameter to generate a list of the available MBeans and related operations and attributes. By default the output is sent to *`instance-name`*`/temp/beanInfo-`*`timestamp`*`.txt`. With a complete list of beans, administrators can write targeted queries and add those queries to larger automated script for monitoring performance and managing an instance. To list the available MBeans and locate AgentManager details, complete the following steps:

1. Start the PAS for OpenEdge instance.
2. Run `oejmx -C` to create a list of available beans.

   Output is sent to the default location, `/temp/beanInfo-`*`timestamp`*`.txt`.
3. Open the output file and search for the `AgentManager` details.

   The AgentManager bean shown, can be used to get agent information, status, and other metrics. For example, the `stopAgent` operations could be used to to stop an agent and `terminateABLSession` could be used to stop a particular session.

```
*************** PASOE:type=OEManager,name=AgentManager
Operation: getAgents(String appName) -->  javax.management.openmbean.CompositeData
Operation: getAgentStatus(String id) -->  javax.management.openmbean.CompositeData
Operation: getAgentMetrics(String id) -->  javax.management.openmbean.CompositeData
Operation: getThreadMetrics(String id) -->  javax.management.openmbean.CompositeData
Operation: getSessionMetrics(String id) -->  javax.management.openmbean.CompositeData
Operation: getConnectionMetrics(String id) -->
javax.management.openmbean.CompositeData
Operation: getRequestMetrics(String id) -->  javax.management.openmbean.CompositeData
Operation: getAgentStacks(String id) -->  javax.management.openmbean.CompositeData
Operation: getSessionStacks(String agentID, String sessionID) -->
javax.management.openmbean.CompositeData
Operation: stopAgent(String agentID, Integer waitToFinish, Integer waitAfterStop)
-->  boolean
Operation: terminateABLSession(String agentID, Integer agentSessionID, Integer
terminateOption) -->  boolean
Operation: trackABLObjects(String agentID, Boolean switch) -->  java.lang.Boolean
Operation: trackingABLObjects(String agentID) -->  boolean
```

Note: Detailed information about available beans can be found by using JConsole. For more information, see the JConsole and JMX topic in *Progress Application Server for OpenEdge: Administration Guide*.

With a list of available beans, you can Write a custom OEJMX query to get agents on page 61 .

Write a custom OEJMX query to get agents

Use OEJMX with the `-Q` *`filepathname`* parameter to use a custom query file. A query file may have multiple queries. Each MBean query is a JSON string having MBean identifier and attribute or operation information. The query format is:

- `O` — ObjectName is a JsonObject
- **`M`** — Operation is a JsonArray
- **`A`** — Attribute is a JsonObject (optional)

In List available MBeans to locate AgentManager details on page 60, steps are provided to list the available beans. From that list, the operations and attributes for the AgentManager are identified. To collect information about the agents, create a custom OEJMX query file using `getAgents` for the default `oepas1` instance. To write a custom OEJMX query, complete these steps:

1. Create a new file containing the following query to access the **AgentManager** bean.

This provides threadpool and session manager information and which can be used to get thread and session information, status, and other metrics.The query getsAgents for the `oepas1` instance.

```
{"O":"PASOE:type=OEManager,name=AgentManager","M":["getAgents","oepas1"]}
```

Note: A query must be a single line containing no embedded spaces. A query file can contain multiple queries. The best practice is to use explicit names over tokens to guarantee the intended results. In this example, `oepas1` replaces the *APP_NAME* token. For more information about using tokens, see OEJMX utility on page 71.

2. Save the file as `getAgents.qry`.

Next you can Get agent information on page 62.

Get agent information

A running PAS for OpenEdge instance must be available. Review Write a custom OEJMX query to get agents on page 61 to get then available agent.

To get agent information, complete the following steps:

1. Change to the bin directory for the instance,*instance-name*/bin
2. Execute the query using `oejmx -Q` *Filepath*`/getAgents.qry`

   The default output location is *CATALINA_TEMP_DIR*`/`*queryname*`-out-`*yyMMdd-HHmm*`.txt`
3. Review the output `getAgents-out`*yyMMdd-HHmm*`.txt` file to see the query response.

   Notice the output shows the original query on line 1 and the results on the next.

```
Query: Line 1. Object: PASOE:type=OEManager,name=AgentManager, Method:
getAgents(oepas1)
Result:
{"getAgents":{"agents":[{"agentId":"QqnRYSLuRjqu0kneMNXAcw","pid":"16952","state":"AVAILABLE"}]}}
```

Note: Using `oepas1` provides the agent pid, `16952` that can be used to gather session metrics.

For more details on the OEJMX utility, see the OEJMX utility on page 71.

Generate a stack trace

OEJMX provides access to the stack information for a session by querying the AgentManager to find active sessions and then querying those sessions to generate a stack trace.

1. Write a query to `getSessionMetrics`.

   Example:

```
{"O":"PASOE:type=OEManager,name=AgentManager","M":["getSessionMetrics","16952"]}
```

Note: The query uses a specific AgentID, `16952` to return session metrics for a specific session.

2. Save the query as `getSessionMetrics.qry`.
3. Run the query using `oejmx -Q /path/getSessionMetrics.qry`.
4. Review the output in *instance-name/temp/*`getSessionMetrics-out-`*yyMMdd-HHmm*.

   Notice that **Session-7** is in **ACTIVE** state and still running some ABL code.

```
{
"getSessionMetrics": {
"AgentSession": [{ "SessionState": "IDLE", "ConnectionId":
"org.codehaus.jettison.json.JSONObject$Null", "SessionMemory": 630630,
 "EndTime": "org.codehaus.jettison.json.JSONObject$Null", "ThreadId": -1,
"SessionExternalState": 0,
"StartTime": "2018-07-30T15:18:28.736", "SessionId": 4 },{ "SessionState": "ACTIVE",
 "ConnectionId": 20,
"SessionMemory": 1646192,  "EndTime": "org.codehaus.jettison.json.JSONObject$Null",
 "ThreadId": 6, "SessionExternalState": 0,
"StartTime": "2018-07-30T15:18:28.736", "SessionId": 7 }
]
}
}
```

5. Gather more information about **"SessionId":7**, run the following getSessionStacks query to get information about individual sessions and identify the running code.

   This collect the getSessionStacks details.

```
{"O":"PASOE:type=OEManager,name=AgentManager","M":["getSessionStacks",
"16952","7"]}
```

6. Save the query as `getSessionStacks.qry`.
7. Run the query using the query `oejmx -Q /path/getSessionStacks.qry`.
8. Review the output in*instance-name/temp/*`getSessionStack-out-`*yyMMdd-HHmm*.

The output displays a list of the stack trace listing files that were in use by the session.

```
{
"getSessionStacks": {
"ABLStacks": [{
"Status": "Executing",
"Databases": "",
"BrokerSessionId": "A8A564D83461D3718EFA71A19C055C0D793DA292F6DF.oepas",
"Callstack": [{ "Line": 50, "Routine": "GetLogistics.p", "Source": "GetLogistics.p"
 }
,{ "Line": 3, "Routine": "BalanceReport.p", "Source": "BalanceReport.p" }
],
"StartupParams": "-pf /usr/Progress/dlc/startup.pf,(end .pf),-logginglevel 2,-logfile

/usr/Progress/wrk/oepas/logs/oepas.agent.log,-uburl
AppServerDC://0.0.0.0:43420/,-logname oepas,
-logentrytypes ASPlumbing,DB.Connects, -ubpropfile
/usr/Progress/wrk/oepas/conf/openedge.properties,
-ASID 1,-ipver IPv4,-sectok XXXXXXXXXXXXXXXXXXXXXX,-T
/usr/Progress/wrk/oepas/temp","Propath": ",
/usr/Progress/wrk/oepas/webapps/ROOT/WEB-INF/openedge,/usr/Progress/wrk/oepas/openedge,
/usr/Progress/dlc/tty,/usr/Progress/dlc/tty/adecomm.pl,/usr/Progress/dlc/tty/adeedit.pl,
/usr/Progress/dlc/tty/adecomp.pl,/usr/Progress/dlc/tty/adeshar.pl,/usr/Progress/dlc/tty/prodict.pl,
/usr/Progress/dlc/tty/OpenEdge.Core.pl,/usr/Progress/dlc/tty/OpenEdge.ServerAdmin.pl,
/usr/Progress/dlc/tty/dataadmin.pl,/usr/Progress/dlc/tty/ablunit.pl,
/usr/Progress/dlc/tty/OpenEdge.BusinessLogic.pl,
/usr/Progress/dlc/tty/OpenEdge.KendoUIBuilder.pl,
/usr/Progress/dlc/tty/netlib/OpenEdge.Net.pl,/usr/Progress/dlc,/usr/Progress/dlc/bin",
"AgentSessionId": 7,
"OO4GLObjs": [{ "ObjectId": "001026", "Class": "OpenEdge/Testing/GetInfo.cls" }
,{ "ObjectId": "STATIC", "Class": "OpenEdge/Testing/APIClass.cls" }
,{ "ObjectId": "STATIC", "Class": "OpenEdge/Testing/GetInfo.cls" }
,{ "ObjectId": "001000", "Class": "OpenEdge/Rest/Admin/AppServerStatus.r" }
],"PersProcs": ""
}
]
}
}
```

From more information see the Common OEJMX queries on page 64 .

Common OEJMX queries

This topic highlights commonly used queries and what information they provide about your application.

Get active session information

When to use: To identify active sessions in order to gather further information about those sessions.

Query:

```
{"O":"PASOE:type=OEManager,name=SessionManager","M":["getMetrics","oepas1"]}
```

Note: In this example, the token *APP_NAME* is replaced with `oepas1`.

Result:

The output provides details about client connections including avgQueueDepth, maxConcurrentClients, maxQueueDepth, reads, requests and timesQueued. The maxQueueDepth can be used to determine if there are any bottlenecks causing the requests to be queued which could indicated areas to improve performance by reducing the queuing of requests.

```
{ "getMetrics": { "accessTime": "2018-06-25T15:43:12.766-04:00", "avgQueueDepth": "0",

 "concurrentConnectedClients":"0",  "maxConcurrentClients": "50", "maxQueueDepth": "0",
 "readErrors": "0",
 "reads": "30133", "requests": "15059", "reserveConnectionTimeouts": "0", "startTime":
 "2018-06-25T15:20:33.981-04:00",
"timesQueued": "0",  "type": "OE_BROKER", "writeErrors": "0", "writes": "30132" } }
```

Get a stack trace for a session

When to use: By supplying a specific *AGENT_ID* and *SESSION_ID*, the query can return a stack trace for the session.

Query:

```
{"O":"PASOE:type=OEManager,name=AgentManager","M":["getSessionStacks","16952","7"]}
```

Note: In this example, the token *AGENT_ID* is replaced with 16952 and *SESSION_ID* is replaced with 7.

Result:

The output displays a list of the stack trace listing files that were in use by the session. Used in conjunction with the GetSessionMetrics to identify active sessions like **SessionID=7**, developers can target specific files for the **ACTIVE** session to investigate.


```
{
"getSessionStacks": {
"ABLStacks": [{
"Status": "Executing",
"Databases": "",
"BrokerSessionId": "A8A564D83461D3718EFA71A19C055C0D793DA292F6DF.oepas",
"Callstack": [{ "Line": 50, "Routine": "GetLogistics.p", "Source": "GetLogistics.p" }
,{ "Line": 3, "Routine": "BalanceReport.p", "Source": "BalanceReport.p" }
],
"StartupParams": "-pf /usr/Progress/dlc/startup.pf,(end .pf),-logginglevel 2,
-logfile /usr/Progress/wrk/oepas/logs/oepas.agent.log,-uburl
AppServerDC://0.0.0.0:43420/,
-logname oepas,-logentrytypes ASPlumbing,DB.Connects,-ubpropfile
/usr/Progress/wrk/oepas/conf/openedge.properties,
-ASID 1,-ipver IPv4,-sectok XXXXXXXXXXXXXXXXXXXXX,-T /usr/Progress/wrk/oepas/temp",
"Propath":
",/usr/Progress/wrk/oepas/webapps/ROOT/WEB-INF/openedge,/usr/Progress/wrk/oepas/openedge,
/usr/Progress/dlc/tty,/usr/Progress/dlc/tty/adecomm.pl,/usr/Progress/dlc/tty/adeedit.pl,/usr/Progress/dlc/tty/adecomp.pl,
/usr/Progress/dlc/tty/adeshar.pl,/usr/Progress/dlc/tty/prodict.pl,/usr/Progress/dlc/tty/OpenEdge.Core.pl,
/usr/Progress/dlc/tty/OpenEdge.ServerAdmin.pl,/usr/Progress/dlc/tty/dataadmin.pl,/usr/Progress/dlc/tty/ablunit.pl,
/usr/Progress/dlc/tty/OpenEdge.BusinessLogic.pl,/usr/Progress/dlc/tty/OpenEdge.KendoUIBuilder.pl,
/usr/Progress/dlc/tty/netlib/OpenEdge.Net.pl,/usr/Progress/dlc,/usr/Progress/dlc/bin","AgentSessionId":
 7,
"OO4GLObjs": [{ "ObjectId": "001026", "Class": "OpenEdge/Testing/GetInfo.cls" }
,{ "ObjectId": "STATIC", "Class": "OpenEdge/Testing/APIClass.cls" }
,{ "ObjectId": "STATIC", "Class": "OpenEdge/Testing/GetInfo.cls" }
,{ "ObjectId": "001000", "Class": "OpenEdge/Rest/Admin/AppServerStatus.r" }
],
"PersProcs": ""
}
]
}}
```


Get metrics for a specific ABL application

When to use: SessionManager metrics provide information about the number of requests that were executed/failed/queued while running on a specific ABL Application.

Query:


```
{"O":"PASOE:type=OEManager,name=SessionManager","M":["getMetrics","oepas1"]
```


Note: In this example, the token *SESSION_ID* is replaced with 7.

Result:

The output demonstrates the number of **requests** and in this example the maxQueueDepth and avgQueueDepth are both 0, which indicates that no clients were ever queued. This indicates that the server is running well.

```
{
  "getMetrics": {
 "accessTime": "2018-06-25T15:43:12.766-04:00",
 "avgQueueDepth": "0",
 "concurrentConnectedClients":"0",
 "maxConcurrentClients": "50",
 "maxQueueDepth": "0",
 "readErrors": "0",
 "reads": "30133",
 "requests": "15059",
 "reserveConnectionTimeouts": "0",
 "startTime": "2018-06-25T15:20:33.981-04:00",
 "timesQueued": "0",
 "type": "OE_BROKER",
 "writeErrors": "0",
 "writes": "30132"
    }
}
```

Get hung client connections for a specific ABL application

When to use: HungClient Connections provides information about the number of requests that are still running in a given ABL Application to identify clients connections that are hung.

Query:

```
{"O":"PASOE:type=OEManager,name=SessionManager","M":["getHungClientConnections","oepas1",6000]}
```

Note: In this example, the token *APP_NAME* is replaced with `oepas1`.

Result:

The `elapsedTimeMs` can be used to help identify long running requests which could indicated a hung client connection.


```
{
   "getHungClientConnections":[
      {
         "httpSessionId":"",
         "reqStartTimeStr":"2018-06-25T15:32:48.139-0400",
         "clientName":"172.16.21.106",
         "executerThreadId":"thd-5",
         "requestID":"3fj1RO35XYk/FDkZSN5UUw",
         "requestProcedure":"retrieveAccnts.p",
         "requestUrl":"http://localhost:29402/apsv",
         "sessionID":"C4F5EA614E9B1F8D94E28D7A87185DA923C4EE22F204.oepas1",
         "adapterType":"APSV",
         "elapsedTimeMs":35298
      },
      {
         "httpSessionId":"",
         "reqStartTimeStr":"2018-06-25T15:32:47.667-0400",
         "clientName":"localhost",
         "executerThreadId":"thd-1",
         "requestID":"rw4IDagih4Q/FDkZAMm8TQ",
         "requestProcedure":"mycustomer.p",
         "requestUrl":"http://localhost:29402/apsv",
         "sessionID":"9E0DDCB1CF61036B5D52587FA41CDEAAE9C8D8713B59.oepas1",
         "adapterType":"APSV",
         "elapsedTimeMs":35771
      },
      {
         "httpSessionId":"",
         "reqStartTimeStr":"2018-06-25T15:33:11.131-0400",
         "clientName":"localhost",
         "executerThreadId":"thd-1",
         "requestID":"gBR4KKyWS6E/FDoZuBeYZQ",
         "requestProcedure":"getlist.p",
         "requestUrl":"http://localhost:29402/apsv",
         "sessionID":"13512413B5FE9737F0180B7439EF7921EA0CDBA52753.oepas1",
         "adapterType":"APSV",
         "elapsedTimeMs":12307
      }
   ]
}
```


Get information about active session on a given agent

When to use:

To get agent session metrics including state of the session, memory consumed, session-Id and time started among others.

Query:


```
{"O":"PASOE:type=OEManager,name=AgentManager","M":["getAgentMetrics","16952"]}
```


Note: In this example, the token *AGENT_ID* is replaced with `16952`.

Result:

Returns session metrics associated with the agent.

```
{
    "getAgentMetrics": {
        "AgentStatHist": [{
            "ActiveThreads": 1,
            "OpenConnections": 1,
            "ExitedSessions": 0,
            "ExitedThreads": 0,
            "CStackMemory": 2097152,
            "ClosedConnections": 0,
            "OverheadMemory": -7015299910500,
            "ActiveSessions": 2
        }]
    }
}
```

Get agent session metrics

When to use:

Agent Session Metrics provides state of the session, memory consumed, session-Id, time it was started for each agent session.

Query:

```
{"O":"PASOE:type=OEManager,name=AgentManager","M":["getSessionMetrics","16952"]}
```

Note: In this example, the token *AGENT_ID* is replaced with `16952`.

Result:

The output demonstrates information for all sessions (**4 & 7**). Information includes SessionState, ConnectionId, SessionMemory, EndTime, ThreadId, SessionExternalState, StartTime and SesssionId.

```
{
    "getSessionMetrics": {
        "AgentSession": [{
            "SessionState": "IDLE",
            "ConnectionId": "org.codehaus.jettison.json.JSONObject$Null",
            "SessionMemory": -2341561411224,
            "EndTime": "org.codehaus.jettison.json.JSONObject$Null",
            "ThreadId": -1,
            "SessionExternalState": 0,
            "StartTime": "2018-07-11T13:33:45.491",
            "SessionId": 4
        }, {
            "SessionState": "IDLE",
            "ConnectionId": "org.codehaus.jettison.json.JSONObject$Null",
            "SessionMemory": -7916428289816,
            "EndTime": "org.codehaus.jettison.json.JSONObject$Null",
            "ThreadId": -1,
            "SessionExternalState": 0,
            "StartTime": "2018-07-11T13:33:45.491",
            "SessionId": 7
        }]
    }
}
```

Get Active threads from Tomcat thread pool

When to use:

Gets information about the active threads for a given Tomcat thread pool

Query:

```
# Tokens: APP_NAME, AGENT_ID, AGENT_PID, SESSION_ID
# Get Active threads from Tomcat thread Pool
{"O":"Catalina:type=Executor,name=tomcatThreadPool","A":"activeCount"}
```

Note: For more information about using tokens, see OEJMX utility on page 71.

Result:

Returns **10** as activeCount.

```
{"activeCount":10}
```

Get APSV Transport metrics for the given ABLApp and WebApp

When to use:

Gets the metrics for the requests that were executed on the APSV transport. It gives the count on number of requests that were connected/disconnected, failing and many more.

Query:

```
# Tokens: APP_NAME, AGENT_ID, AGENT_PID, SESSION_ID
# Get APSV Transport metrics for the given ABLApp and WebApp
{"O":"PASOE:type=OEManager,name=ApsvTransportManager","M":["getMetrics","oepas1","ROOT"]}
```

Note: For more information about using tokens, seeOEJMX utility on page 71.

Result:

Returns the APSV Transport Metrics for a given OEABL WebApp.

```
{
    "getMetrics": {
        "disconnectRequests": "15",
        "connectErrors": "0",
        "disconnectErrors": "0",
        "sessionErrors": "0",
        "startTime": "2018-07-11T08:17:35.329-04:00",
        "sessionRequests": "39",
        "forbiddenErrors": "0",
        "type": "OE_APSV_TRANSPORT",
        "connectRequests": "15",
        "accessTime": "2018-07-11T13:11:48.554-04:00",
        "statusRequests": "0"
    }
}
```

OEJMX utility

OEJMX is a command-line utility you can use to:

- List all the MBeans, including their attributes and their operations for a PAS for OpenEdge instance running on a local machine.
- Run MBean queries.

Output from OEJMX is written to a text file. The script for OEJMX is located in the *CATALINA_BASE*/bin directory. This directory is typically included in your PATH environment variable.

Syntax

```
oejmx [ [-C ] | [ -Q queryfile ]  [ -R ] ] [-O outputfile]
```

Parameters

`-C`

Write Mbean information, including attributes and their operations, to the output file specified by the `-O` parameter.

`-Q` *queryfile*

Run queries from a file located in *queryfile* and write results to the output file specified by the `-O` parameter. If no query file is supplied, a default file located in *CATALINA_BASE*/bin/jmxqueries/default.qry is used.

Each MBean query is a JSON string having an MBean identifier and attribute or operation information. A query file may have multiple queries, with one query per line.

Attribute query:

```
{"O":"MBean-identifier","A":"Attribute-name"}
```

Operation (java method):{"O":"<MBean identifier>","M":["Operation",<parameter1 value>,<parameter2 value>…]}

There are a few tokens which could be used instead of PASOE parameter values:

- APP_NAMEAGENT_ID
- AGENT_PID
- SESSION_ID
- ABL_SESSION_ID

When the query processes, the token is replaced with its current value. If there are multiple values of a token a separate query is processed for each one of them.

`-R`

Specifies that only query results are written to a results file.

`-O outputfile`

Specifies the results file path, which is where MBean information or query information is written.

The default value for Mbean information is *CATALINA_TEMP_DIR*/beanInfo-TIMESTAMP:*yyMMdd-HHmm*.txt.

The default value for queries is *CATALINA_TEMP_DIR*/*queryname*-out-TIMESTAMP:*yyMMdd-HHmm*.txt

configutil

Exports the source installation configurations and imports them to a different OpenEdge installation on the same or different machine.

When you install OpenEdge, the configuration information is stored in separate configuration files. This utility helps you to capture all installation configuration information and uses it to automatically configure another OpenEdge installation on the same or a different machine.

This command-line utility is available as configutil.bat file for Windows and as a shell script for UNIX in the <install-dir>/bin location. You can run the utility from the proenv shell.

Operating system	Syntax
UNIX Windows	configutil *operation* [*option*] *operation*: export [*option*] import -backupfile *backup zip path* undoimport -importbackupfile *import backup zip path* help [export | import | undoimport] Use help for detailed usage

The configutil utility does not support the following:

- Exporting from Openedge 64-bit and importing into Openedge 32-bit and vice-versa
- Exporting from a Windows machine and importing into a UNIX machine and vice-versa
- Exporting property settings containing non-ascii characters

Configutil Operations

After installing OpenEdge, you may have customized the configuration files, such as adding new servers, changing ports, and changing logging levels. All these customizations are typically stored in some configuration files such as .properties, .xml, and .pf files, or in the Window's registry. The Configutil utility allows you to capture these configuration changes and preserve them so that they can be applied to a different OpenEdge installation. This process of capturing the existing configuration setting is referred to as the export operation. This process creates a zip file named export<timestamp>.zip.

The configutil utility allows you to apply the exported zip file on another OpenEdge 11.7 installation such that all your configuration settings from the earlier OpenEdge 11.7 installation are applied to the new installation. This process is the import operation.

After an import, if you want to revert to your original configuration, the configutil utility allows you to do so using the undoimport option.

Configutil Export

Exports all the settings stored in configuration files and registry keys that are specified in the configutil.properties file in the <install-dir>/install/configbackup location. You can create a custom.properties file to contain only the list of files and registry entries that you want to export and use the -propFile option to specify the custom file. Here is the syntax for the export operation:

```
Export Usage: configutil export [option]
The options are as follows:
  -backupdir  The directory where the exported zip file will be stored
                        Defaults to $WRKDIR/backup
  -propFile   The user defined configutil.properties file
                           Defaults to $DLC/install/configbackup/configutil.properties
```

The configutil export creates the export<time_stamp>.zip file that contains the following files:

- The `configuration.xml` file that contains all the file contents.
- The `OEImport.properties` file that contains a list of files and environment variables.
- The `export_info` file that contains information on the version, operating system, and bit of OpenEdge installation.

Example

```
Proenv> configutil export

Configurations successfully exported at location:
C:\DLCWOR~1\WRK\backup\export161117072437.zip
```

In the example, since `-backupdir` is not provided, the utility will export the settings to `$WRKDIR/backup/export<timestamp>.zip`.

By default, the utility will export all settings to `$DLC/install/configbackup/configutil.properties`. You may specify a different location using `-propFile`.

Here is another example:

```
Proenv> configutil export -backupdir /custom/directory -propFile
/custom/myconfigutil.properties

Configurations successfully exported at location:
/custom/directory/export161117072437.zip
```

Since `-backupdir` is provided, the utility will export the settings to the `/custom/directory/export<time_stamp>.zip` file.

Since `-propFile` is provided, the utility will export all the settings stored in files and registries that are specified in the `myconfigutil.properties` file.

Configutil Import

Imports the settings in the exported zip file and applies it to the new installation. If there is a setting with a different value on the target machine, the import operation overwrites the value contained in the exported setting. If a setting is not present on the source machine but does exist on the target machine, the import operation leaves the target setting as is. In other words, it will not delete the setting from the target machine. Here is the syntax for the import operation:

```
Import Usage: configutil import -backupfile backup zip path

The options are as follows:
  - backupfile The directory path where the exported file (export Timestamp.zip) is
saved on the
machine where you want to import the settings of another OpenEdge installation. The
default path
is WRKDIR/backup.
```

If you commented out a setting in the configuration.properties file, the setting is not overwritten during import.

```
proenv>configutil import -backupfile C:\DLCWOR~1\WRK\backup\export161117072437.zip
```

In the example, the utility will import the settings from the export161117072437.zip file.

Configutil Undoimport

Reverts the import operation. The import option also creates a backup of existing settings of the target machine before the exported settings are applied. The backup is stored as $WRKDIR/backup/import_archive/<Time_Stamp>/import_backup.zip. To undo the import operation, you can use this archive to restore the original configuration. Here is the syntax for the undoimport operation:

```
Undo Import Usage: configutil undoimport -importbackupfile import backup zip path

The options are as follows:
  -importbackupfile The directory path
$WRKDIR/backup/import_archive/<Time_Stamp>/import_backup.zip where the backup file is
saved during the import operation.
```

Configutil.properties

The configutil.properties file is located in the <install-dir>/install/configbackup folder. This file contains a master list of files or registry entries that contain configuration settings that can be exported. Here is a sample configutil.properties file:

```
[Environment]
OVERRIDE_PATHS=false
OVERRIDE_HOSTNAME=false

#Configuration files for Core.
[$DLC/startup.pf]
[$DLC/bin/progress.ini]

#Configuration files for OEE/OEM.
[$DLC/properties/JavaUtilitys.properties]
[$DLC/properties/activemqsCache.xml]
…
# [HKLM:\Software\PSC\PROGRESS\{version}]
```

OVERRIDE_PATHS and OVERRIDE_HOSTNAME

The configutil.properties file contains an Environment section that has `OVERRIDE_PATHS` and `OVERRIDE_HOSTNAME` entries. By default, these are set to false.

While exporting a setting, the utility tries to ascertain if the value is a valid path on the source machine or a value containing the hostname that matches that of the source machine. If you set `OVERRIDE_PATHS` to false, any value which contains valid path on the source machine is not over-written during import. Similarly, if you set `OVERRIDE_HOSTNAME` to false, any value which contains hostname matching that of source machine is not over-written during import. (Note that all values containing hostname do not qualify for `OVERRIDE_HOSTNAME`).

This is done so because path and host name of the source machine may not have any meaning on target machine. If you set these values to true, the utility overwrites all the setting values during import irrespective of whether it is a path or a hostname.

Note: It is recommended that you store your path settings relative to `$DLC` or `$WRKDIR` to avoid merge conflicts during import resulting in path settings not being imported.

If you do not want to export any specific files, you can comment the entries for them in the `configutil.properties` file.

FORCE_OVERRIDE

If you want to always overwrite some specific setting during import, you can use the FORCE_OVERRIDE option. For example, there can be a path value that you may want to overwrite during import even if the OVERRIDE_PATHS entry is set to false. This option can be specified in the configutil.properties file as shown below:

```
[Setting Name]
FORCE_OVERRIDE:(PropertyHierarchy).propertyName
```

The format of PropertyHierarchy is:

```
ParentSection/ChildSection[/ChildSection/...])
```

How to override a setting in the startup.pf file

If you have set OVERRIDE_PATHNAME=false, but want to overwrite a single setting in startup.pf such as "-k c:\test\foo.txt" in the startup.pf file, you must add the following in the configutil.properties file:

```
[$DLC/startup.pf]
FORCE_OVERRIDE:.-k
```

DONT_OVERRIDE

If you want to never overwrite some specific settings during import, such as an appserver's broker port value, you can use the DONT_OVERRIDE option.

This option can be specified in the configutil.properties file as shown below:

```
[Setting Name]
DONT_OVERRIDE:(PropertyHierarchy).propertyName
```

The format of PropertyHierarchy is:

```
ParentSection/ChildSection[/ChildSection/...])
```

How to override a setting in the ubroker.properties file

If you do not want to override the `srvrLogFile` and the `brkrLogFile` properties in the `[UBroker.AS.asbroker1]` section, add the following in the `configutil.properties` file:

```
[$DLC/properties/ubroker.properties]
   DONT_OVERRIDE:(UBroker/AS/asbroker1).srvrLogFile
   DONT_OVERRIDE:(UBroker/AS/asbroker1).brokerLogFile
```

If you do not want to override the `srvrLogFile` and the `brkrLogFile` properties in the `[UBroker.AS]` section, add the following in the `configutil.properties` file:

```
[$DLC/properties/ubroker.properties]
  DONT_OVERRIDE:(UBroker/AS).srvrLogFile
  DONT_OVERRIDE:(UBroker/AS).brokerLogFile
```

If you do not want to override the `srvrLogFile` property in all subsections of `[UBroker.AS]` such as `[UBroker.AS.asbroker1]`, `[UBroker.AS.asbroker2]`, add the following in the `configutil.properties` file:

```
[$DLC/properties/ubroker.properties]
   DONT_OVERRIDE:(UBroker/AS/*).srvrLogFile
   DONT_OVERRIDE:(UBroker/AS/*).brokerLogFile
```

If you have set `OVERRIDE_HOSTNAME=false` and you want to forcibly override the `restURL` property in the `[REST.restmgr1]` section, add the following in the `configutil.properties` file:

```
[$DLC/properties/ubroker.properties]
  FORCE_OVERRIDE: (REST/restmgr1).restUrl
```

For properties that do not belongs to any section, override as follows:

```
DONT_OVERRIDE:.propertyName
FORCE_OVERRIDE:.propertyName
```

Note: For more examples on registry entries and XML files, see the Importing Window's registry and Importing XML files sections.

Exporting and importing registry entries based settings

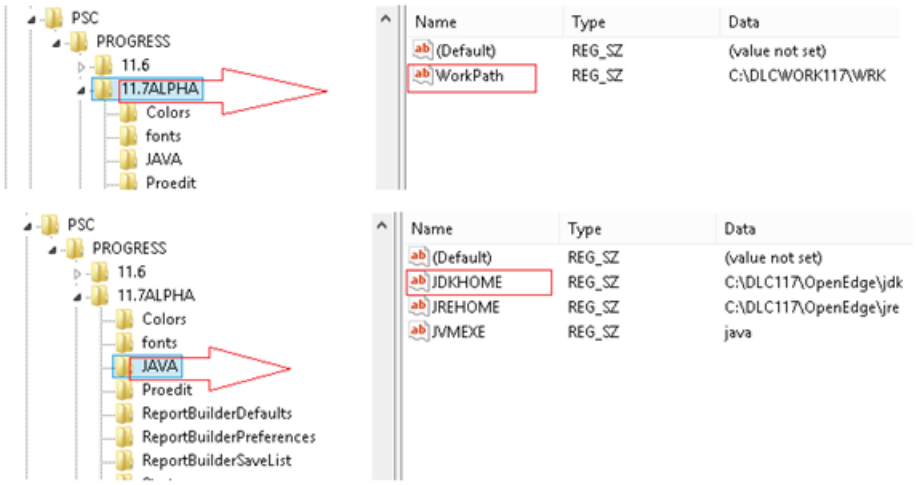
By default, the configutil.properties file does not contain registry entries for export. You must manually add the registry path and the entries to the $DLC/install/configbackup/configutil.properties file as follows:

```
[HKLM:\Software\PSC\PROGRESS\{version}]
```

Here are the short forms of the registry hives:

```
• HKCR -  HKEY_CLASSES_ROOT.
• HKCU -  HKEY_CURRENT_USER.
• HKLM -  HKEY_LOCAL_MACHINE.
• HKU  -  HKEY_USERS.
• HKCC -  HKEY_CURRENT_CONFIG.
• HKPD -  HKEY_PERFORMANCE_DATA.
```

How to override a setting the Window's registry:



As in the above image of the registry entries, if you want to forcibly overwrite WorkPath which is contained at the level of 11.7ALPHA node (node 0) and do not want to overwrite JDKHOME under JAVA node (which is a child node), add the following in the configutil.properties file to set DONT_OVERRIDE or FORCE_OVERRIDE entries:

```
[HKCU:\SOFTWARE\PSC\PROGRESS\{version}]
FORCE_OVERRIDE:.WorkPath
DONT_OVERRIDE:(/JAVA) JDKHOME
```

Exporting and Importing XML files-based settings

XML files always have their own hierarchy. If you want to forcibly over-write or do not want to over-write a setting stored in an XML file, you must create its hierarchy to refer to the element's attributes and value.

If you want to create a hierarchy rule for an element's text, then use the following format:

```
(<ElementHierachy>).#text
```

If you want to create a hierarchy rule for an element's attribute, then use the following format:

```
(<ElementHierachy>).<attributeName>
```

<ElementHierarcy> has the following format:

```
(<RootElement[ID]>/<childElement[ID]/[<childElement[ID]/...>)
```

ID is optional for each root or child element and, if not provided, it applies to all the elements. Here is a sample XML file:

```
<web-app>
    <servlet>
        <servlet-name>Aia</servlet-name>
        <servlet-class>com.progress.aia.Aia</servlet-class>
        <init-param>
           <param-name>InstallDir</param-name>
           <param-value>C:\Progress\OpenEdge</param-value>
        </init-param>
        <init-param>
           <param-name>instanceName</param-name>
           <param-value>Aia1</param-value>
        </init-param>
        <init-param>
           <param-name>propertyFileName</param-name>
          <param-value>C:\Progress\OpenEdge/properties/ubroker.properties</param-value>

        </init-param>
   </servlet>
</web-app>
```

If you do not want to OVERRIDE or FORCE-OVERRIDE the highlighted init-param, you can construct the hierarchy rule as follows:

```
(web-app/servlet/init-param[InstallDir]/param-value).#text
```

Here, web-app is the RootElement. Its child is servlet, and the servlet contains init-param. The init-param is the text value of installDir. So, installDir is the ID that you provide within square brackets.

Note that an ID for servlet is not added, so the above hierarchy applies to all servlets that have init-param as InstallDir. If you want the rule to be applied to a single servlet, say AIA, your hierarchy rule must look like this:

```
(web-app/servlet[Aia]/init-param[InstallDir]/param-value).#text
```

Here is a sample configutil.properties with XML hierarchy rule:

```
[$TCIB_HOME/webapps/aia/WEB-INF/web.xml]
  FORCE_OVERRIDE:(web-app/servlet/init-param[InstallDir]/param-value).#text
  FORCE_OVERRIDE:(web-app/servlet/init-param[propertyFileName]/param-value).#text
```

Here is another sample XML file:

```
<portlist udp="" udpmin="1" udpmax="300" tcp="" tcpmin="1" tcpmax="200">
   <port number="21" type="TCP" name="FTP"/>
   <port number="80" type="TCP" name="HTTP"/>
   <port number="110" type="TCP" name="POP3"/>
   <port number="25" type="TCP" name="SMTP"/>
   <port number="119" type="TCP" name="NNTP"/>
   <port number="143" type="TCP" name="IMAP"/>
   <port number="23" type="TCP" name="Telnet"/>
   <port number="389" type="TCP" name="LDAP"/>
   <port number="547" type="TCP" name="DHCPv6"/>
   <port number="53" type="UDP" name="DNS"/>
   <port number="161" type="UDP" name="SNMP"/>
 </portlist>
```

If you want to retain or override the number for HTTP port, add the following in the configutil.properties file:

```
[$oemgmt/conf/Ports.xml]
DONT_OVERRIDE:(portlist/port[HTTP]).number
```

Logging of import and export operations

The details of export and import operations are logged in the `$WRKDIR/oeie.log` file.

The logging settings are configured in the `log4j.properties` file in the `$DLC/install/configbackup` location.

3

Working with Instances

Instances are a standard Apache Tomcat feature. They allow you to create individual deployment and/or development servers that share the core Progress Application Server that you installed.

Instances are independently running copies of the core Progress Application Server. Each instance runs on its own JVM, has its own configuration with unique ports, and hosts its own web applications. However, each instance runs a Tomcat server that uses a number of common files from the same `$CATALINA_HOME` directory. Each instance has an alias. The default value is the directory in which the instance was created, but it can be set to some other string.

As a best practice, Progress recommends that you deploy your web applications to an instance of the Progress Application Server, rather than deploying to the Progress Application Server that you installed. This practice prevents accidental corruption of the core executables, configuration settings, and libraries. It also prevents accidental deletion of web applications if the core Progress Application Server is removed during an OpenEdge uninstall.

For details, see the following topics:

- Overview of instances in PAS for OpenEdge
- Instance management with TCMAN
- Creating instances with TCMAN
- Installing and running an instance as a UNIX daemon
- Installing and running an instance as a Windows service
- Automate lifecycle event scripts

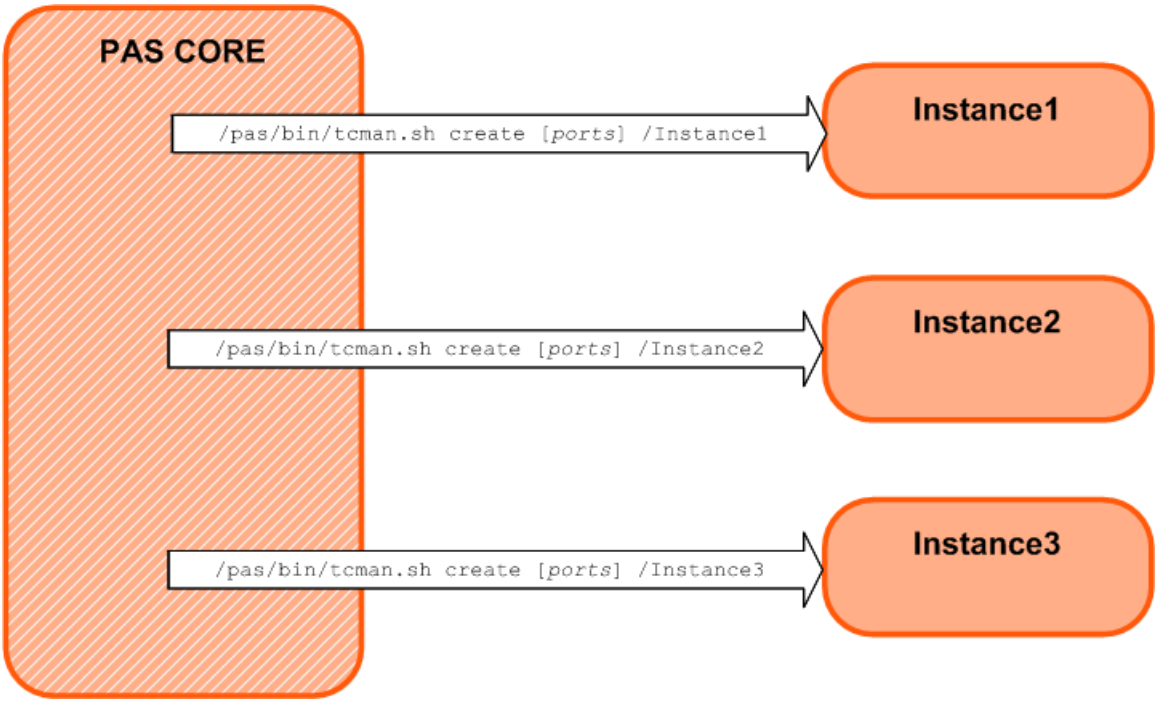
Overview of instances in PAS for OpenEdge

After you install the core Progress Application Server, you can create an instance.

Instances are a standard Apache Tomcat feature. They allow you to create individual deployment and/or development servers that are based on the core Progress Application Server that you installed.

The following figure illustrates the creation of multiple instances using the TCMAN command-line utility (with syntax simplified).

Figure 1: Generating PAS instances



Instances are independently running copies of the core Progress Application Server. Each instance runs on its own JVM, has its own configuration with unique ports, and hosts its own web applications. However, each instance runs a Tomcat server that uses a number of common files from the same `$CATALINA_HOME` directory.

There are a number of advantages when you deploy your web applications to an instance of the Progress Application Server, rather than deploying to the Progress Application Server that you installed. This practice prevents accidental corruption of the core executables, configuration settings, and libraries. It also prevents accidental deletion of web applications if the core Progress Application Server is removed when you uninstall a Progress PAS product.

Some additional advantages of instances are:

- Updates to the core Apache Tomcat server libraries and executables do not affect your web applications. You avoid the necessity of updating the applications and/or re-configuring them.
- You can establish different security policies for each of the instances.
- You can tailor the JVM for individual applications, since each instance runs in its own JVM with its own configuration.
- Instances provide you with quick way to create a test server for experimenting with new configurations and applications without the danger of permanently corrupting an existing server.
- You can package an instance as a Web application and deploy it to other PAS core servers.

You use $CATALINA_HOME/bin/tcman.sh create command to create a new instance.

When you create an instance, the root directory of the instance is assigned to the CATALINA_BASE environment variable within the scripts in its /bin directory. The root directory of the installed (core) Progress Application Server is assigned to the CATALINA_HOME environment variable in the scripts in the instance's /bin directory. (Notice that the scope of these environment variables is limited to the context of an individual instance's /binscripts.)

All instances of a core Progress Application Server execute a set of common JAR files, scripts, and libraries from the following directories on the parent server:

- $CATALINA_HOME/lib
- $CATALINA_HOME/common/lib
- $CATALINA_HOME/bin

However, each instance is created with :

- A $CATALINA_BASE/bin/ directory with its own copy of some of the scripts from the core PAS. These include scripts for start up, shut down, deployment, running TCMAN actions, and so on.
- A $CATALINA_BASE/conf/ directory with its own copy of properties and configuration files.
- A $CATALINA_BASE/webapps/ which initially only contains the ROOT Web application.
- A number of directories that are initially empty. These include /logs, /temp, /work, and /common/lib.

Instance management with TCMAN

TCMAN includes actions for configuring, starting, stopping, monitoring, and deleting instances.

The following table is a brief description of the instance management actions that you can perform with TCMAN. Entries link to the reference topics that provide more details, syntax, and examples.

Action	Purpose
create	Create an instance of the Progress Application Server.
delete	Remove the directory tree and all of the files in an instance.
start	Start an instance of a Progress Application Server.
stop	Stop a running instance.
config	View, add, update, or delete the property values specified in ../conf/appserver.properties.
test	Displays information on the configuration and environment of an instance. It also displays information about error conditions.
instances	Display all the instances created from the Progress Application Server installed in $CATALINA_HOME.
unregister	Stop tracking an instance by removing the instance's entry from the $CATALINA_HOME/conf/instances.[unix|windows] file.

Action	Purpose
register	Register an instance for tracking purposes. (Note that instances are registered for tracking by default when they are created. The register action is only necessary if you explicitly unregistered an instance.)
clean	Truncate, move, or delete the log files located in the `/logs` directory of either the core server or an instance.
version	Show the Apache Tomcat runtime version and OS information for an instance.

Creating instances with TCMAN

Before you can create an instance of the Progress Application Server (PAS) using the TCMAN command line utility, you must:

- Install the core Progress Application Server
- Install JDK or JRE version 1.7 or later
- Set the `JAVA_HOME` or `JRE_HOME` environment variable to the JDK/JRE install directory .

Note: TCMAN is a Progress extension of the basic Tomcat administrative utilities. TCMAN simplifies instance creation and management.

An instance runs the Tomcat executable of a core PAS, but it runs in a separate JVM, is configured with its own unique ports, and other properties. (You should not attempt to use the installed, core PAS as a development or production server.) Instances allow you to run a variety of server configurations without corrupting the files in the core server. They also allow you to update the core server without re-deploying or re-configuring your Web applications.

To create an instance using the TCMAN utility:

1. Open a command shell and navigate to `$CATALINA_HOME/bin`.

   `$CATALINA_HOME` is the directory where you installed the core Progress Application Server.

2. Run `tcman.sh create` *`basepath`* (or `tcman.bat` on Windows systems) .

   The *`base_path`* parameter specifies the path name where you will create the instance. It is the only required parameter for the `create` action. If you are creating multiple running instances, you should override the default port assignments by specifying the following parameters:

–p port_num	Specify the TCP port that listens for HTTP messages. The default is 8080.
–P port_num	Specify the TCP port that listens for HTTPS messages. The default is 8443.

   You can also activate these ports:

–s port_num	Specify the TCP port to use to stop an instance. (Required on Windows systems, optional on UNIX)

–j port_num	Specify the TCP port that listens for AJP13 messages, an Apache protocol for handling requests from a web server to an application server. (Optional on both Windows and UNIX systems)

See Create an instance (create) on page 217 for information about other parameters.

3. (Optional) Deploy remote management applications from `$CATALINA_HOME/extras` to the instance.

   Remote management applications are not pre-installed, and installing them is a security decision. For example, you might want to eliminate access to the configuration and control of instances by not deploying management applications to production servers, while deploying management applications to development servers.

   To deploy a management application:

   a) Open a command shell and navigate to `$CATALINA_BASE/bin`.

   b) Run `tcman.sh deploy '$CATALINA_HOME/extras/admin_webapp.war'`.

   The *admin_webapp*`.war` can be one of the following:

`host-manager.war`	A Tomcat administration application used to get server information and provide other functionality. It should not be necessary to deploy `host-manager.war` if you are using the TCMAN utilities.
`manager.war`	A Tomcat administration application which you must deploy in order to run some TCMAN actions. See the TCMAN Reference material for information on which TCMAN actions require deployment of `manager.war`.
Progress applications	Progress products can have web applications that enable the use of their own administrative tools.

For example the following command line creates an instance of `/psc/pashome` in `/psc/acme1` and specifies its ports:

```
$: /psc/pashome/bin/tcman.sh create -p 8501 -P 8601 -s 8701 /psc/acme1
Server instance acme1 created at /psc/acme1
```

See also

Server actions on page 216
Create an instance (create) on page 217
Delete an instance (delete) on page 219
Display and manage an instance's configuration (config) on page 220
Display or modify the server features of an instance (feature) on page 222
Clean up or archive server log files (clean) on page 224
Display server instances (instances) on page 224
Register an instance for tracking (register) on page 227

Stop tracking an instance (unregister) on page 236
Start an instance (start) on page 231
Stop an instance (stop) on page 233
Display server, OS, and runtime version information (version) on page 234
Test a server configuration (test) on page 235

Installing and running an instance as a UNIX daemon

A daemon process (called a service on Windows systems) is an application without a user interface that runs in the background and responds to requests. Web servers like PAS and Tomcat typically run as Windows services or UNIX daemons.

Note: If you run a PAS instance with `tcman.sh start`, the instance runs in the context of the command shell process. It is not available as a system service that can handle external client requests. The instance must be installed as a daemon process before you can run it as a functioning Web server.

The file `$CATALINA_HOME/bin/daemon.sh` can be used as a template for starting Tomcat automatically at boot time as a child of the `init` process. For more information, see:

https://tomcat.apache.org/tomcat-7.0-doc/setup.html#Unix_daemon

However, you will need to consult with a system administrator before you can configure and run PAS as a daemon process due to differences among UNIX systems and because you need administrative privileges for access to the system.

Installing and running an instance as a Windows service

To install a Progress Application Server (PAS) instance as a Windows service, you must have administrator privileges. On systems with User Account Control (UAC), you must disable UAC as well.

A service (called a daemon process on UNIX systems) is an application without a user interface that runs in the background and provides core operating system functionality. Web servers like PAS and Tomcat typically run as Windows services or UNIX daemons.

Note: If you run a PAS instance with the TCMAN `start` action, the instance runs in the context of the command shell process. It is not available as a system service that can handle external client requests. The instance must be registered as a Window service before you can start it as a service.

This is a summary of how to register and run a PAS instance as a Windows service:

1. Open a command prompt window.
2. Navigate to the core PAS `/bin` directory (`$CATALINA_HOME/bin`).
3. Run the TCMAN `service` action specifying an instance name and the `register` parameter . For example:

```
tcman service oepas1 register
```

where `oepas1` is the name of the default instance created when you installed PAS for OpenEdge.

4. Run the TCMAN `service` action specifying an instance name and the `start` parameter. For example:

```
tcman service oepas1 start
```

Note: You can also use the TCMAN `service` action to check the running status, stop, and unregister a PAS for OpenEdge instance as a Windows service. You can also use the Windows Microsoft Management Console (MMC) or the `sc config` command to start, stop, and check the status of a service.

See also

Register and manage an instance as a Windows service (service) on page 228

Automate lifecycle event scripts

Tomcat servers have the ability to fire lifecycle events. These events include the startup and shutdown of a server. Administrators can write scripts to automatically run when these events occur. For example, scripts can start a third party server when an instance starts. Logically, another script would shut down the third party server when the instance shuts down. The scripts must be located in the *instance*`/bin` directory and use the following naming conventions.

Table 7: Automated lifecycle scripts

Script	Lifecycle event
scriptPrefix`_starting.bat|sh`	Executes when the Tomcat container is started but **before** any web application context are created.
scriptPrefix`_started.bat|sh`	Executes when the Tomcat container is started and **after** all web application contexts are created.
scriptPrefix`_stopping.bat|sh`	Executes **before** stopping an instance
scriptPrefix`_stopped.bat|sh`	Executes **after** stopping an instance.

Note: Replace the *scriptPrefix* with a unique name for each instance.

Scripts execute inline

The startup and shutdown event scripts execute in-line which means the Tomcat startup/shutdown process waits for the event script to complete before continuing. If a script hangs it prevents the lifecycle event from completing.

Logs and errors

The standard server logs, located in the *instance*`/logs` directory, tracks when the scripts are called. If errors are raised by the script, those errors do **not** interrupt the standard server startup or shutdown. Administrators

are responsible for catching errors in their scripts and writing those errors to a script specific log file in the *instance*/logs directory.

4

Application deployment and management

For details, see the following topics:

- Web application management with TCMAN
- OpenEdge Web application deployment
- OpenEdge ABL applications
- Update PROPATH in a production instance with zero downtime
- Passing ABL error objects to REST clients

Web application management with TCMAN

TCMAN includes a number of actions for deploying and managing Web applications to instances.

The following table is a brief description of the Web application management actions that you can perform with TCMAN. Entries link to the reference topics that provide more details, syntax, and examples.

Action	Purpose
list	Display all the Web applications that are deployed on an instance.
info	Display server and OS information for a running instance.
deploy	Deploy a Web application (`.war` file) to a Progress Application Server instance whether the server is running (online) or is not running (offline).

Action	Purpose
undeploy	Remove a Web application from running (online) or stopped (offline) instances.
reload	Restart a deployed, running web application so that the application can pick up changes to its classes or libraries.
status	List information from the core server's memory, including web application statistics.
leaks	List Web applications with potential memory leaks.
enable	Start a Web application that is deployed but not running.
disable	Stop a running Web application.
resources	List the global resources used by the core server.
sessions	Display how many sessions are active for the specified Web application, categorized by their duration.

OpenEdge Web application deployment

The `deployREST.sh` and `deploySOAP.sh` scripts allow you to add REST or SOAP services to a deployed OpenEdge ABL Web application (`oeabl.war`). When you invoke either of the scripts, the OpenEdge ABL Web application must be running. After you deploy the service, you must stop and restart the Progress Application Server instance to run the new service.

REST service deployment (deployREST.sh)

Purpose

Deploys REST services.

Syntax

```
$CATALINA_BASE/bin/deployREST.sh source_descriptor service_name
```

Parameters

source_descriptor

Specify the path of the source descriptor, which can be either a PAAR file containing the descriptor for the REST service or a ZIP file containing Mobile catalog files (or other static files).

service_name

Specify the target service name.

Example

```
$CATALINA_BASE/bin/deployREST.sh CustomerService.paar ROOT
```

SOAP service deployment (deploySOAP.sh)

Purpose

Deploy SOAP services.

Syntax

```
$CATALINA_BASE/bin/deploySOAP.sh source_descriptor service_name
```

Parameters

source_descriptor

Specify the path of the source descriptor, which is a WSM file.

service_name

Specify the target service name.

Example

```
$CATALINA_BASE/bin/deploySOAP.sh CustomerService.wsm ROOT
```

Packaging web applications

The Progress Application Server supports a number of different ways for packaging web applications. Web applications can be packaged as:

- a `.war` file
- a `.zip` file that includes the necessary directory structure (see below for more information)
- a UNIX `.tar` file that includes the necessary directory structure
- a deep copy of the web application's directory
- a soft link (symbolic link) to the web application's directory

The standard directory structure for a web application is:

- `/exampleapp` - The root directory for the application is located in the server's `/webapps` directory (default) or another location configured by the administrator. The root directory contains the `/WEB-INF` folder and any files that have to be visible to the client browser, like HTML and JSP pages. These files can be organized into a subdirectory for more complicated applications.
- `/exampleapp/WEB-INF` - In addition to the directories listed below, this folder contains the Web Application Deployment Descriptor (`web.xml`), which contains the configuration information for the application in a standard format
- `/exampleapp/WEB-INF/classes` - This folder contains any Java class files that are not organized into `.jar` files, including servlet classes.
- `/exampleapp/WEB-INF/lib` - This folder contains any `.jar` libraries needed for the application.

Note: The minimum directory structure required for a web application includes the `/exampleapp/WEB-INF` directory and the `web.xml` file. The `/WEB-INF/classes` and `/lib` directories are only required when distributing loose Java class files and `.jar` files.

For more information on Tomcat directory structure, see http://tomcat.apache.org/tomcat-7.0-doc/index.html.

OpenEdge ABL applications

An ABL application in the context of a PAS for OpenEdge server is a business application having:

- A unique name which can be referenced from administrative tools
- A set of r-code files that comprise the application
- A single PROPATH value that is used to access the applications's r-code
- One or more OpenEdge database connections
- A set of start-up options

One or more ABL applications can execute within the boundaries of a single PAS for OpenEdge Session Manager and the mult-session agent[s] that it controls.

Note: For additional PAS for OpenEdge documentation, including white papers on advanced topics, see:

https://community.progress.com/community_groups/openedge_development/m/documents

---

Update PROPATH in a production instance with zero downtime

PAS for OpenEdge system administrators can dynamically update their applications by enabling live PROPATH updates in production instances.

This feature allows system administrators to:

- Deploy new application r-code to a running server instance.
- Change the PROPATH by using the `oeprop` utility or by deploying an updated application to include the new r-code location.

PAS for OpenEdge ABL sessions that are not currently handling requests then begin executing the new application code. PAS for OpenEdge ABL sessions currently running requests finish those requests before using the new application code.

When the PROPATH is updated, the ABL Sessions execute `sessionShutdownProc`, restart, and then execute `sessionStartupProc`. BoundSessions are not updated with the new PROPATH until the sessions finish executing the bound client requests.

To enable live PROPATH updates in production instances, perform the following steps:

1. Run the proenv utilty.
2. Navigate to the target *instance-name*`/bin` directory.
3. Use the OEPROP utility to set `allowRuntimeUpdates` to `1` in the `openedge.properties` file. For example:

   ```
   oeprop.[sh|bat] AppServer.allowRuntimeUpdates=1
   ```

   This property turns on a process that checks the `openedge.properties` file for changes and interactively passes those changes to the PAS for OpenEdge multi-session agents.
4. Manually restart your PAS for OpenEdge instance to have this change take effect.

Passing ABL error objects to REST clients

When a runtime error (such as an application error, system error, or stop condition) occurs in ABL code, an error object containing error details is passed to ABL clients automatically. However, by default, HTTP clients (such as browsers), that access a REST service deployed on PAS for OpenEdge web application, receive only a brief JSON error message similar to the following example:

```
{
 "_retVal":""
 "_errors":[{
             "_errorMsg":"ERROR condition: procUserAppError: my error message (7211)",
             "_errorNum":1
           }]
}
```

The error message may not be useful because it does not contain details such as error type, stack trace, etc. You can change this default behavior as follows.

- **To configure the web application to send error details**: Open the `runtime.props` file located in *`OpenEdge_work_directory`*`\WRK\`*`PAS_instance`*`\webapps\`*`webapp_name`*`\WEB-INF\adapters\rest` and change the `serviceFaultLevel` setting from 2 to **4**.

  For example:

  ```
  <bpm:serviceFaultLevel>4</bpm:serviceFaultLevel>
  ```

- **To include the ABL call stack in the error details**: Add an `-errorstack` parameter to the `agentStartupParam` property in the `openedge.properties` file located in *`OpenEdge_work_directory`*`\WRK\`*`PAS_instance`*`\conf`.

  For example:

  ```
  agentStartupParam=-T "${catalina.base}/temp" -errorstack
  ```

After these properties are set, a detailed JSON error message is generated. Here is an example:

```
{
"_retVal":""
"_errors":[
              {

               "error_info":{

                                "oe_version":"12.0.0",

                                "errorMsg":"ERROR condition: procUserAppError: my error
message (7211)",

                                "ablCallStack":"procUserAppError server1.p at line 65
(C:\OpenEdge\120\WRK\pas1\webapps\ROOT\WEB-INF\openedge\server1.p)",

                                "numMessages":1,

                                "severity":0,

                                "isAppError":true,

                                "returnValue":"",

                                "messages":[

                                {

                                              "message":"procUserAppError: my error
message",

                                              "messageNum":1

                                }
                                ]

               },

              "_errorMsg":"ERROR condition: procUserAppError: my error message (7211)",


               "_errorNum":1

               }
],

}
```

You can also log these error messages on the server side. To do this, edit the `logging.xml` file in the web application's `WEB-INF` directory, uncomment the following property, and set the logging level to `TRACE`.

```
<logger name="com.progress.appserv.endpoints.oe.Endpoint" level="TRACE"/>
```

5

HTTP sessions

HTTP sessions are an industry standard for sharing context information including user identity and user-specific data during multiple request/response interactions between a client and a web server. This context information is required for load balancing. While SOAP, REST and WEB transports are implicitly using HTTP sessions, the APSV transport adds this functionality through a combination of properties that are enabled by default for the ROOT application.

For details, see the following topics:

- Overview of HTTP sessions
- Clusters and sticky sessions

Overview of HTTP sessions

HTTP sessions is an industry standard feature that allows Web servers to maintain user identity and to store user-specific data during multiple request/response interactions between a client application and a Web application. HTTP sessions preserves:

- Information about the session itself (session identifier, creation time, time last accessed, etc.)
- Contextual information about the user (client login state, for example, plus whatever else the Web application needs to save)

The Tomcat servlet container, which is built into PAS for OpenEdge, provides HTTP sessions as the default method for preserving session and contextual information.

Since information in HTTP sessions can be accessible to a cluster of PAS for OpenEdge instances, HTTP sessions are useful for load balancing. (See Load balancing options for PAS for OpenEdge on page 101.)

Note: Information about the third party technologies mentioned in this section is very general. For more detailed information, do a Web search for key terms: HTTP sessions, JSESSIONID,

Administrative considerations

To enable HTTP sessions for the APSV transport (HTTP session is automatically enabled for SOAP, REST and WEB), both `enableRequestChunking` and `useHTTPSessions` must be activated in the `/conf/openedge.properties` file. For example:

```
enableRequestChunking=1
useHTTPSessions=1
```

HTTP sessions are enabled by default in PAS for OpenEdge instances and there is nothing you need do to configure them. However, you can check these settings in an instance's `openedge.properties` file to ensure that HTTP sessions are enabled.

Note: If you want to use HTTP sessions, the ABL client must be OpenEdge 11.6 or a later release. Open Clients (.NET and JAVA) are not supported.

Notice that chunking must be enabled as well as HTTP sessions. Chunking is an efficient method for sending a large amount of data in a request or response when the size of the content is not known. Chunking also that all the data gets transmitted in a single post operation in the context of a single session to a single server. You can use chunking if HTTP sessions is disabled (if, for example, you only have one instance and do not require load balancing . However, you will get a startup error if chunking is disabled and HTTP sessions is enabled.

When load balancing is implemented using clusters of PAS for OpenEdge instances, HTTP sessions allows session context information to be available to all the nodes in the cluster. Therefore a single session between a client and a Web application can be maintained over any number of instances in the cluster. See Clusters and sticky sessions on page 99 and Tomcat load balancing on page 105for more information.

Note: You can use the TCMAN `sessions` action to display how many sessions are active for a Web application.

Programmatic considerations

The Web application determines when to start and stop a session. Also, It not only determines *what* contextual information will be stored in the session, it determines *how* that information will be stored.

Note: The Spring Security framework built in to PAS for OpenEdge saves client login state information to HTTP sessions.

By default, when a Web application running on a PAS for OpenEdge instance decides to initiate a session with a client, it creates a JSESSIONID cookie that identifies the session. The cookie is sent to the client. Every time the client runs a request to that application, the cookie gets sent back to the PAS for OpenEdge instance, whose job is to identify and retrieve the stored data that is associated with the session id.

Context information required for the session is automatically saved in HTTP sessions. Again, it is up to the application to determine what needs to be saved and how to save it. Context data itself can be saved to and retrieved from HTTP sessions. Or you can save the context information externally and just save a reference to it in HTTP sessions. The latter option is more efficient when there is a large amount of data associated with the session.

In ABL for example, the SERVER-CONNECTION-CONTEXT attribute of the SESSION system handle is commonly used to set context information. If the context information can be expressed as a string of less than 32K, the attribute can be set to that value. The 32K maximum may be even less, depending on the other limitations imposed by your particular configuration. However, the more data you store in SERVER-CONNECTION-CONTEXT the more performance is likely to suffer. Therefore, the recommended practice is to store the context information in a database and use the SERVER-CONNECTION-CONTEXT attribute as a reference to the context information.

The implementation of how to store context varies widely, and depends on application requirements. For more information on how to use the SERVER-CONNECTION-CONTEXT attribute see *Progress Application Server for OpenEdge: Application and Migration Development Guide* and *OpenEdge Development: ABL Reference*.

See also

Display Web application HTTP sessions (sessions) on page 216

Clusters and sticky sessions

HTTP sessions is an enabling technology for load balancing. With HTTP sessions, session information can be shared with any PAS for OpenEdge instance in a defined cluster of instances. Therefore, any instance in the cluster can handle subsequent client requests within the correct context.

Instances in a cluster must all be created from a common core PAS for OpenEdge installation. ($CATALINA_HOME). To add an instance to a cluster, you must turn on the cluster property in the `/conf/server.xml` file of the instance. To avoid corrupting `server.xml` use the TCMAN utility to update the property. For example:

```
/bin/tcman.sh feature Cluster=on
```

Sticky sessions

HTTP sessions can be used to contain context information whether sticky sessions are enabled or disabled.

Sticky sessions, another industry standard for Web servers, binds a session to a particular instance. In other words, after an intial client request is satisfied by a Web application running in a PAS for OpenEdge instance, subsequent requests are routed to the same Web application running on the same instance. You would want sticky sessions enabled when:

- The Web application is storing state information about the session
- The Web application launches persistent procedures, since the memory space that persistent procedures run in cannot be shared by multiple instances.

In a load balancing configuration, sticky sessions are usually enabled or disabled by configuring the Web server. In an Apache Web server, you enable sticky sessions by adding a `route` property to `BalancerMember` and setting the `stickeysession` property. For example:

```
#Load balancer Proxy settings (sticky sessions)
      <Proxy "balancer://mylbgroup">
      BalancerMember "http://172.16.21.39:11600" route=oepas1
      BalancerMember "http://172.16.21.39:11610" route=oepas2
      BalancerMember "http://172.16.21.39:11620" route=oepas3
      </Proxy>
      ProxyPass "/test" "balancer://mylbgroup"
      stickysession=JSESSIONID|jsessionid nofailover=On
      ProxyPassReverse "/test" "balancer://mylbgroup"
```

When sticky sessions are not enabled on an Apache Web server, the instances in the configuration are specified by their IP addresses only and the `stickysession` property is not set. For example:

```
#Load balancer Proxy settings (session free)
      <Proxy "balancer://mylbgroup">
      BalancerMember "http://172.16.21.39:11600"
      BalancerMember "http://172.16.21.39:11610"
      BalancerMember "http://172.16.21.39:11620"
      </Proxy>
      ProxyPass "/test" "balancer://mylbgroup"
      ProxyPassReverse "/test" "balancer://mylbgroup"
```

Note:

- The Tomcat servlet container in PAS for OpenEdge does not allow Web applications to change a session to a sticky session.
- Sticky sessions can have a negative impact on performance in a load balancing configuration since client requests can be bound to a single node among all available nodes.

6

Load balancing options for PAS for OpenEdge

If you are familiar with configuring classic WebSpeed, you will find that load balancing (distributing client requests among multiple servers) is quite different in PAS for OpenEdge. In PAS for OpenEdge, there is no NameServer or `ubroker.properties` file to implement load balancing. Instead, load balancing is accomplished through industry standard HTTP options.

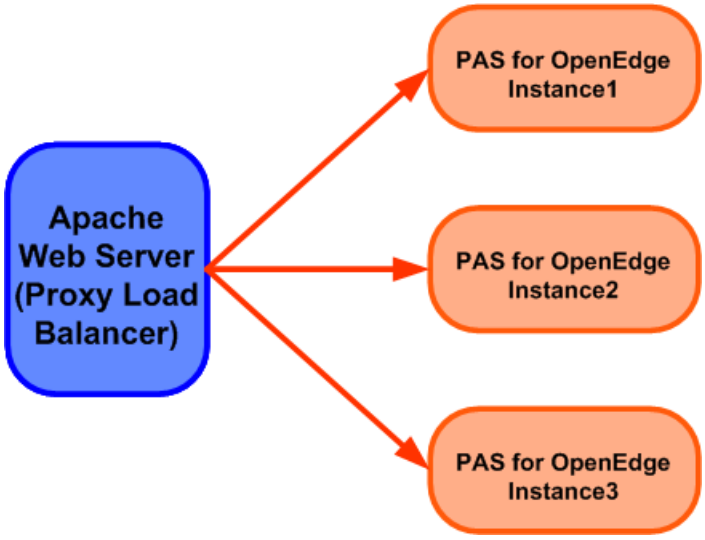
This section describes three of the most popular load balancing options for PAS for OpenEdge. It includes an overview of configuration tasks for each option, and, where applicable, describes the strengths and weaknesses of the option. There are many more 3rd party load balancing products available. It is not possible to recommend the best load balancing option for your application

For details, see the following topics:

- Apache proxy host load balancing
- Tomcat load balancing
- Amazon load balancing

Apache proxy host load balancing

With Apache proxy host load balancing, an Apache HTTP Server (also known as Apache) directs requests to a particular PAS for OpenEdge instance. The following figure illustrates this load balancing configuration.



Apache works as a proxy re-directing HTTP requests to a URL that is unique to each instance. This method requires no configuration of the PAS for OpenEdge instance, all of the load balancing is done by the Apache server. The available PAS for OpenEdge instances are defined in an *apache_install_dir*`/conf/httpd.conf` file.

See also: http://httpd.apache.org/docs/current/mod/mod_proxy.html

Note: The disadvantage in this type of load balancing is the Apache server does not monitor the running status of instances. If an instance crashes or is otherwise unavailable, the Apache load balancer continues to try to send requests to the unavailable instance and requests will be lost. Compare with Tomcat load balancing on page 105, where the running status of instances is monitored.

See also

Requirements for Apache proxy load balancing on page 102
Configuring the Apache server for proxy load balancing on page 103

Requirements for Apache proxy load balancing

The requirements for implementing Apache proxy load balancing are:

1. **A properly configured Apache HTTP Server.**

   You configure a load balancing Apache HTTP Server (commonly referred to as *Apache*) with a list of available PAS for OpenEdge instances. In addition, specify whether you want subsequent requests to return to the same instance (session-managed, also known *sticky sessions* in Apache documentation), or to any available instance (session-free).

   If you are running one or more web application available to the public, install Apache in a DMZ with the PAS for OpenEdge instances running behind a second firewall.

2. **Properly configured PAS for OpenEdge instances.**

   Configuration of instances is contingent on whether sticky sessions are enabled on the Apache load balancer. All the instances must be configured to match the session type of the load balancer.

See also

Configuring the Apache server for proxy load balancing on page 103
Configuring PAS for OpenEdge instances for proxy load balancing on page 104

Configuring the Apache server for proxy load balancing

Configuring an Apache HTTP Server to implement load balancing involves updating the *apache_install_dir*/conf/httpd.conf file to:

- Specify the proxy modules to load
- Reference the PAS for OpenEdge instances
- Enable or disable sticky sessions

Note: In addition to configuring the Apache server, you must configure each PAS for OpenEdge instance. See Configuring PAS for OpenEdge instances for proxy load balancing on page 104

Loading proxy modules

Proxy modules set up the Apache web server as a proxy to handle requests to multiple PAS for OpenEdge instances. The modules are loaded by un-commenting `LoadModule` directives in the `httpd.conf` file of the Apache web server. The `LoadModule` directives include:

- `LoadModule proxy_module modules/mod_proxy.so`
- `LoadModule proxy_connect_module modules/mod_proxy_connect.so`
- `LoadModule proxy_http_module modules/mod_proxy_http.so`
- `LoadModule slotmem_shm_module modules/mod_slotmem_shm.so`
- `LoadModule bmethod_byrequests_module modules/mod_lbmethod_byrequests.so`

Note: Depending on which version of the Apache server you are using, there may be additional modules required. See the Apache web site (http://httpd.apache.org/) for documentation on your version of the Apache server.

Referencing instances and configuring session types

Update Apache's `httpd.conf` file to include a load balancer section that lists the PAS for OpenEdge instances in the load balancing group. This can be configured two different ways:

- **Session free** — any request can be sent to any PAS for OpenEdge instance.
- **Sticky session** — once a session is started all requests return to the same PAS for OpenEdge instance until the session is terminated.

If there is no need for client requests to return the the same PAS for OpenEdge instance (i.e. maintaining server-side context is not required), your entries in `httpd.conf` should resemble the following example:

```
#Load balancer Proxy settings (session free)
      <Proxy "balancer://mylbgroup">
      BalancerMember "http://172.16.21.39:11600"
      BalancerMember "http://172.16.21.39:11610"
      BalancerMember "http://172.16.21.39:11620"
      </Proxy>
      ProxyPass "/test" "balancer://mylbgroup"
      ProxyPassReverse "/test" "balancer://mylbgroup"
```

Notice that, in the example above, the instances where Apache can refer requests are grouped as `//mylbgroup` and the URL of these instances (the `BalancerMember` properties) are specified.

If it is necessary for sequential requests to go to the same instance, as they are for the APSV transport, you enable sticky sessions by adding a `route` property to `BalancerMember` and by enabling JSESSIONID cookies. For example:

```
#Load balancer Proxy settings (sticky sessions)
      <Proxy "balancer://mylbgroup">
      BalancerMember "http://172.16.21.39:11600" route=oepas1
      BalancerMember "http://172.16.21.39:11610" route=oepas2
      BalancerMember "http://172.16.21.39:11620" route=oepas3
      </Proxy>
      ProxyPass "/test" "balancer://mylbgroup" stickysession=JSESSIONID|jsessionid
nofailover=On
      ProxyPassReverse "/test" "balancer://mylbgroup"
```

In the example above notice that:

- The `route` property must match the `jvmroute` value in the `server.xml` file of the instance.
- The `stickysession` property enables the JSESSIONID cookie, which is the mechanism used to identify the session.

Note: Sticky sessions are required for the APSV transport, but not for WEB, REST, or SOAP.

Note: For more information about session configurations in the `httpd.conf` file, see the Apache web site (http://httpd.apache.org/) for documentation on your version of the Apache server.

See also

Configuring PAS for OpenEdge instances for proxy load balancing on page 104

Configuring PAS for OpenEdge instances for proxy load balancing

To configure PAS for OpenEdge instances for proxy load balancing, verify that chunking and HTTP sessions are enabled. These properties are enabled by default for the SOAP, REST, and WEB transports. See Use HTTP sessions for instructions on how to enable chunking and HTTP sessions for the APSV transport.

If you are not enabling sticky sessions on the Apache server, you must configure the Apache server for session-free load balancing in which subsequent requests do not return to the same instance. Perform the following steps:

1. Enable `Cluster` property in the `/conf/server.xml` file for each instance. You can use the `feature` action of the PASMAN command-line utility to set this property. For example:

```
pasman.[sh|bat] -I instance-name feature Cluster=on
```

2. Set the `jvmRoute` alias to blank. The `jvmRoute` alias is controlled by the `psc.as.alias` value and can be set to blank with PASMAN (rather than by manually editing `/conf/server.xml`). For example:

```
pasman.[sh|bat] config psc.as.alias=
```

Note: The `jvmRoute` alias translates to the route value in the `http.conf` file of the Apache server. See Configuring the Apache server for proxy load balancing on page 103 for more information.

3. Open the *instance-name*`/conf/context.xml` file in each instance and comment out the `Manager` section. For example:

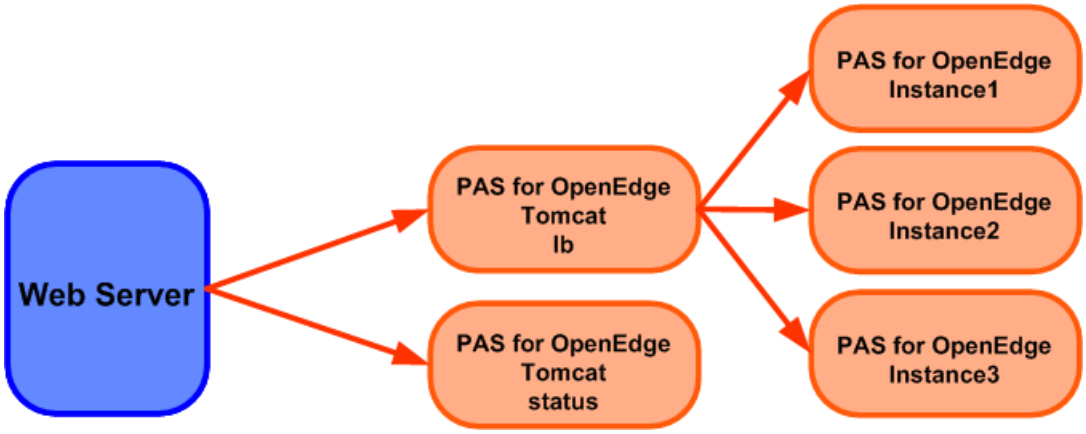
```
<!--  <Manager
        maxActiveSessions="-1"
        pathname=""
        processExpiresFrequency="6"
        maxInactiveInterval="${psc.as.session.timeout}"
        sessionIdLength="22" >
    </Manager>  -->
```

See also

Configuring the Apache server for proxy load balancing on page 103

Tomcat load balancing

Tomcat load balancing is achieved by designating a PAS for OpenEdge instance that only refers requests from a Web server to instances that can process them. Tomcat is built into PAS for OpenEdge. The following figure illustrates this load balancing configuration.



Some advantages of this configuration, compared to Apache proxy host load balancing on page 101, are:

- The PAS for OpenEdge instance that is designated as a load balancer (`lb` in the figure above) is aware of the runtime status of each instance in its group and will not forward a request to an instance that is down.
- Tomcat load balancing allows you to configure a PAS for OpenEdge instance as as a monitor of runtime statistics for each instance in the load balancing group (the `status` instance in the figure above).

See also

Requirements for Tomcat load balancing on page 106
Configuring PAS for OpenEdge for Tomcat load balancing on page 107

Requirements for Tomcat load balancing

The following are the requirements to implement Tomcat load balancing:

1. **A properly configured Web server.**

   If you are running one or more web application available to the public, install the Web server in a DMZ with the PAS for OpenEdge instances running behind a second firewall.

2. **A PAS for OpenEdge instance that is dedicated to load balancing.**

   You should not deploy any web applications to the load balancer.

3. **A `workers.properties` file.**

   The `workers.properties` file specifies the existence of a load balancer, the workers (instances) that are available to the load balancer as servers that can handle requests, an optional status worker that monitors the runtime status of the other workers.

   You can create a `workers.properties` file with the TCMAN `workers` action.

See also

Create a Tomcat worker configuration file (workers) on page 237

Configuring an Apache web server for Tomcat load balancing

Apache web server configuration for Tomcat load balancing involves enabling the AJP13 protocol for communicating with the load balancer and adding a number of definitions that reference the load balancer.

Note: Tomcat load balancing can be done with web servers other than Apache. However, only Apache configuration is described here because Apache is arguably the most common web server, and it is not practicable to describe the configuration details of all the available web servers.

Perform the following steps:

1. Open the Apache `/conf/httpd.conf` file and un-comment the`LoadModule` directives in the `httpd.conf` file that enable AJP13. The `LoadModule` directives include:

   - `LoadModule proxy_ajp_module modules/mod_proxy_ajp.so`
   - `LoadModule proxy_balancer_module modules/mod_proxy_balancer.so`

   Note: Depending on which version of the Apache server you are using, there may be additional modules required. See the Apache web site (http://httpd.apache.org/) for documentation on your version of the Apache server.

2. Add the following definitions to `/conf/httpd.conf`:

```
#workers.properties load balancing config

      LoadModule jk_module apache_install_dir/modules/mod_jk.so
      JkWorkersFile apache_install_dir/conf/workers.properties
      JkShmFile apache_install_dir/logs/mod_jk.shm
      JkLogFile apache_install_dir/logs/mod_jk.log
      JkLogLevel info
      JkLogStampFormat "[%a %b %d %H:%M:%S %Y]"
      JkMount /* jklb
```

3. Use PASMAN to enable the AJP13 port in the `/conf/server.xml` file.

   For example:

```
pasman.[sh|bat] -I instance-name feature AJP13=on
```

Apache now passes all requests to a PAS for OpenEdge load balancing instance via its AJP13 port.

Note: The PAS for OpenEdge load balancing instance is referenced in the example above as `jklb` (`JkMount /* jklb`). The actual instance name must be `lb`.

Configuring PAS for OpenEdge for Tomcat load balancing

Configuring PAS for OpenEdge to enable Tomcat load balancing involves:

1. **Creating an instance (`lb`) that is dedicated to load balancing.**

   You can use the PASMAN `create` action to instantiate an instance that must be named either `lb` or `jklb`. No other configuration is required, but note that you should not deploy any web applications to the `lb` instance.

2. **Optionally creating an instance (`status`) to gather metrics on the instances in the load balancing group.**

   You can use the PASMAN `create` action to instantiate an instance that must be named `status` or `jkstatus`. No other configuration is required, but note that you should not deploy any web applications to the `status` instance.

   Also note that the `lb` automatically keeps track of the runtime status of instances in its load balancing group. It will not forward a request to an instance that has crashed or is not running for some other reason. Therefore, the status instance is only required if you want to monitor performance data about the instances in the load balancing group.

3. **Generating and deploying a `worker.properties` file.**

   The `worker.properties` file is a Tomcat feature that defines the load balancing instance (`lb`), the status instance , and the instances that handle requests `status`.

   You can create a preliminary `worker.properties` file using the PASMAN`workers` action. The action gathers information on all instances registered to the local PAS for OpenEdge core server (CATALINA_HOME).

   Note: There must be an instance named `lb` or `jklb` that will function as the load balancer.

   The `workers` action only includes local instances. If you have instances on multiple machines, you must:

   1. Run the PASMAN `workers` action on each instance.
   2. Combine the `workers.properties` file from each machine into a single `workers.properties` file.
   3. Comment out the `workers.common.host` property.
   4. Add a `workers.`*`instance_name`*`.host=`*`host_name`* property to each instance.

   Note:

   - The PASMAN `workers` action references the content of the `$CATALINA_HOME/extras/workers.template` file when it builds a preliminary `worker.properties` file. You can modify the `workers.template` file to include settings that are typical in your local configuration.
   - The preliminary `worker.properties` is created in the `temp` directory of the CATALINA_HOME server (for example, `$DLC/servers/pasoe/temp`). Use this file as a starting point; you may need to modify parameter values to better suit your particular application configuration.

See also

Create an instance (create) on page 217
Create a Tomcat worker configuration file (workers) on page 237

Amazon load balancing

Amazon Web Services offers an Elastic Load Balancing (https://aws.amazon.com/elasticloadbalancing/) product that distributes incoming application across multiple Amazon EC2 instances in the cloud. It also offers an Auto Scaling (https://aws.amazon.com/documentation/autoscaling/) product that automatically monitors the EC2 instances and manages them according to policies that you can define. You can use these products together to create a cloud-based load balancing configuration for PAS for OpenEdge.

See also

Configuring Elastic Load Balancing and Auto Scaling on page 109

Configuring Elastic Load Balancing and Auto Scaling

This topic is an overview of configuring Amazon Elastic Load Balancing and Auto Scaling as it applies to PAS for OpenEdge. For more information about the Amazon products, see https://aws.amazon.com/documentation/elastic-load-balancing/ and https://aws.amazon.com/documentation/autoscaling/.

The following table lists the relevant terms applied to the components of these products.

Table 8: Elastic Load Balancing and Auto Scaling components

Component	Description
Virtual Private Cloud (VPC)	A private network of your application machines, which is your internal, "safe" network away from the internet. Access can be limited to specific incoming IP addresses or ranges.
Amazon Machine Image (AMI)	A copy of a virtual machine that, when launched, will automatically start the processes that support the load balanced application.
Elastic Load Balancer (ELB)	An internet facing load balancer that redirects work to the instances running in a VPC. Monitors instance health and redirects using a proprietary cookie (`AWS_ELB`).
Launch Configuration	A definition of the configuration of the AMI you want to launch automatically when scaling up.
Auto Scaling Groups	A definition of a group of AMI's that specifies their ELB and the scaling policy (how and when scaling up or down occurs).

The basic steps for creating an Elastic Load Balancing and Auto Scaling configuration are:

1. Create a VPC (or use the default) which includes:
   - defining a network mask range for your private network
   - defining a security group that limits access to the private network
2. Create the AMIs.

   Each AMI must have a PAS for OpenEdge instance installed, and the instance must be configured to start when the operating system launches. Note that the instances can be configured to use the same ports since they are running on different hosts and therefore will have unique IP addresses in the VPC.
3. Create an ELB, enabling:
   - an internet facing scheme
   - port forwarding (for example, forward port 80 to the PAS for OpenEdge port 8810)
   - sticky sessions (using the `AWS_ELB` cookie) or cookie timeout.
   - cross-zone load balancing
   - log timing
   - idle timeout
   - a Health Check
4. Create a Launch Configuration that specifies:
   - an AMI
   - the instance size (CPUs, memory, and disk space)
   - the VPC
   - the security group
5. Create an Auto Scaling Group that specifies:
   - the ELB
   - the VPC
   - the maximum and minimum number of copys of the AMI that can be launched
   - a Health Check configuration
   - the scaling policy (what metric threshold causes scaling up or down)
   - the tag to apply at instance launch

Note: The preceding steps are just an overview of the procedures required to implement Amazon load balancing. See the Amazon documentation for more complete information.

7

Server security

This section provides information about security configurations for instances of PAS for OpenEdge. Generally speaking, server security support includes standard Tomcat server functionality, the Spring Security Framework, plus OpenEdge security features and customizations.

For details, see the following topics:

- Production instance security customizations
- HTTPS support
- Use SSL/TLS in PAS for OpenEdge
- The Spring Security Framework in PAS for OpenEdge
- JVM security manager
- Manage users and remote access
- Remote access filters

Production instance security customizations

Some of the security customizations on a production instance of PAS for OpenEdge are:

- **`manager` and `host-manager` web applications are undeployed by default**—these are default Tomcat applications that enable remote online administration. These applications are archived in the `$CATALINA_HOME/extras` directory and can be redeployed to enable their functionality.
- **Replacement of the default Tomcat `webapps/ROOT` application**—replaced with a ROOT application that specifically supports Progress applications including application security. The Tomcat ROOT application is archived in the `$CATALINA_HOME/extras` directory.
- **Auto-deployment turned off by default**—prevents the deployment of WAR files that are maliciously or erroneously copied to the instance. If turned on, the instance automatically deploys any new or updated WAR files in its web application directory.

  Note: PAS for OpenEdge automatically unpacks WAR files when they are legitimately deployed using PASMAN or standard Tomcat utilities.

- **Shutdown port disabled for UNIX and Windows**—prevents unauthorized stopping of the instance.

  Note: A shutdown port is optional for instances running on UNIX systems. However, you must specify a shutdown port when you create an instance that runs on Windows systems. The PASMAN utility supports specifying shutdown ports with the `-s` option to the `create` action.

- **Disabled JMX remote access**—although PAS for OpenEdge includes JMX and JConsole support for instance management, the default is for local access only.
- **Web crawler filtering enabled**—prevents instances from being overloaded by sessions initiated by web crawlers.
- **UNIX file system customizations**—file permissions are initially only accessible and executable by ROOT users and groups. This is described in User and file permissions.

HTTPS support

PAS for OpenEdge allows you to configure an HTTPS port when you create an instance of a PAS server. The PASMAN utility supports specifying HTTPS ports with the `-P` option to the `create` action.

For example, the following instance is created with an HTTP port of `8501` and an HTTPS port of `8601`:

```
$: pasman.[sh|bin] create -p 8501 -P 8601 /psc/instance-name
Server instance acme1 created at /psc/acme1
```

In PAS (and in Tomcat as well) HTTPS port support is provided by the Secure Socket Layer (SSL) or Transport Layer Security (TLS) implementation of the Java Secure Socket Extension (JSSE).

Important:

On a production server, you *must* replace the default SSL/TLS Server Certificate with a fully qualified certificate from a Certificate Authority (CA).

Server access via HTTPS is dependent on having an SSL/TLS Server Certificate. To support initial configuration and testing, PSC includes a self-signed server certificate in a Java keystore. (A key store is basically a file that contains the encrypted keys required for encoding and decoding information.) The PSC keystore is formatted according to the PKCS12 format and is maintained using the Java Keytool utility.

However, the default server certificate should only be used on a non-production server, or as a temporary certificate for setting up SSL or TLS on a production server. Using the default certificate on a fully functional and deployed production server, is a serious security risk.

See Use SSL/TLS in PAS for OpenEdge on page 113 for more information.

Use SSL/TLS in PAS for OpenEdge

In PAS for OpenEdge and in Tomcat, HTTPS port support is provided by the Secure Socket Layer (SSL) or Transport Layer Security (TLS) implementation of the Java Secure Socket Extension (JSSE).

TLS is a standard protocol used to implement the encryption required for HTTPS communication.

Note:

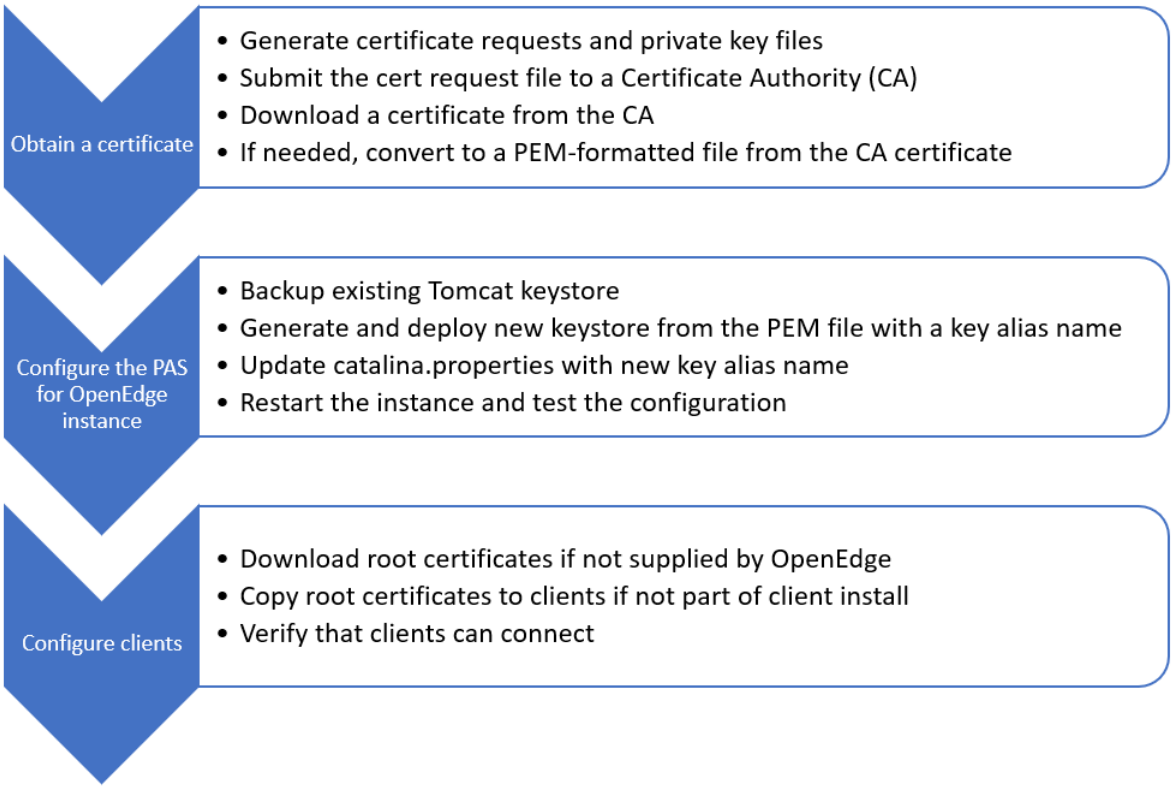
TLS evolved from and superseded SSL. TLS is considered to be more efficient and secure than the older SSL. Even though TLS and SSL are separate and distinct protocols, the PAS for OpenEdge configuration process is the same for both. Therefore we use the term *SSL/TLS*.

Also note that most browsers concurrently support some version of both protocols.

PAS for OpenEdge allows you to specify an HTTPS port when you create an instance. The PASMAN utility supports specifying HTTPS ports with the `-P` option of the `create` action. For example, the following instance is created with an HTTP port of `8501` and an HTTPS port of `8601`:

```
$: pasman.[sh|bin] create -p 8501 -P 8601 /psc/instance-name
Server instance acme1 created at /psc/acme1
```

The following is an overview of SSL/TLS configuration in PAS for OpenEdge:



Important:

On a production instance, you *must* replace the default SSL/TLS Server Certificate shipped with PAS for OpenEdge with a fully qualified certificate from a Certificate Authority (CA).

Instance access via HTTPS is dependent on having an SSL/TLS Server Certificate. To support initial configuration and testing, Progress includes a self-signed server certificate in a Java keystore. (A key store is a file that contains the encrypted keys required for encoding and decoding information.) The Progress Software keystore is formatted according to the PKCS12 format and is maintained using the Java Keytool utility.

However, the default server certificate should only be used on a non-production instance, or as a temporary certificate for setting up SSL or TLS on a production instance. Using the default certificate on a fully functional and deployed production instance, is a serious security risk.

See the following topics for more details about SSL/TLS configuration:

- Obtain a certificate from a Certificate Authority
- Configure a PAS for OpenEdge instance for SSL/TLS
- Configure clients for SSL/TLS

Obtain a certificate from a Certificate Authority

This topic describes how to generate a certificate request file that you can submit to a third party Certificate Authority (CA) to obtain an SSL/TLS certificate. It also explains how to import the certificate from the CA to a PAS for OpenEdge instance.

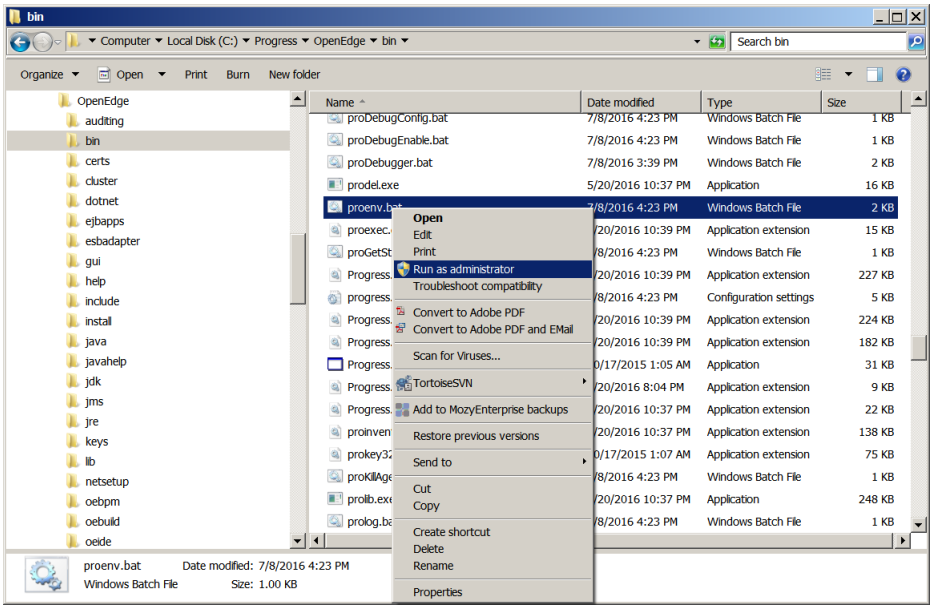
1. Start PROENV and change the directory to `*OpenEdge-Install-Dir*\keys\requests`.

For example:

```
proenv> cd C:\Progress\OpenEdge\keys\requests
```

Note:

On Windows, your must start PROENV with the `Run as administrator` option. For example:



2. Generate a 2048-bit private key (*certificate_name*`.pk1`) and a public key (*certificate_name*`.pk10`) file.

   For example:

```
proenv>pkiutil -keysize 2048 -newreq MyCert
```

Note: PKIUTIL can generate key sizes of 512, 1024, or 2048 bits.

PKIUTIL prompts for a PEM pass phrase (which you must compose) with output similar to the following:

```
Loading 'screen' into random state - done
Generating a 2048 bit RSA private key
..+++
.....................+++
writing new private key to'C:\\dlc/keys/requests/MyCert.pk1'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
----
```

After you enter and verify the pass phrase, the private key file (`MyCert.pk1` in this example) is generated.

Important: Make a note of the PEM pass phrase. It will be required when you import the certificate returned from a CA to create a keystore in Step 5 on page 116.

PKIUTIL prompts for the specific name of the system (Server DNS name) and other information that will be included in the certificate request. The information is incorporated into a Distinguished Name (DN). If a single period '.' is entered, the field will be left blank. For example:

```
-----
Country Name (2 letter code) [US]:US
State or Province Name (full name) []:Massachusetts
Locality Name (eg, city) []:Bedford
Organization Name (eg, company) []:ACME1
Organizational Unit Name (eg, section) []: .
Server DNS name []:bedford.acme1.com
```

A public key file (`MyCert.pk10` in this example) is created. The public key file is what you use to request a new digital certificate from the CA.

3. Submit the public key file (`MyCert.pk10` in this example) to a CA in order to request an SSL certificate.

   The CA returns both a private and a public (or ROOT) SSL certificate. The certificates are files with either a `.crt` or a `.cer` extension. In this example, the files are named `MyCertPriv.cer` and `MyCertPub.cer`

4. When the SSL certificates are received from the CA, copy them to the *OpenEdge-Install-Dir*`\keys\requests` directory.

5. Generate a Privacy Enhance Mail (`.pem`) formatted file from the private SSL certificate (named `MyCertPriv.cer` in these examples) obtained from a CA.

   A PEM file is an encrypted file that contains key store information. You use the OpenEdge PKIUTIL command-line utility to generate the PEM file. You can find more information about the syntax and usage of PKIUTIL in *OpenEdge Getting Started: Installation and Configuration*.

   a) In PROENV, change directory to the PAS for OpenEdge instance's`/conf` directory.

      For example:

      ```
      proenv> cd C:\MyInstance\conf
      ```

   b) Use the `-import` option of PKIUTIL to generate the PEM file from the private SSL certificate.

      For example:

      ```
      proenv> pkiutil -import MyCert OpenEdge-Install-Dir\keys\requests\MyCertPriv.cer
      ```

      Note: `MySSLPrivKey` is the stem filename of the PEM file that will be generated from `MyCertPriv.cer`.

   c) When prompted enter the password you used when you created the keystore (i.e. the `.pk1` file) in Step 2 on page 115.

   PKIUTIL creates a file with a `.pem` extension (`MySSLPrivKey.pem` in this example) in the *OpenEdge-Install-Dir*`\keys\` directory.

Configure a PAS for OpenEdge instance for SSL/TLS

Configuring a PAS for OpenEdge instance to use SSL/TLS involves updating the default Tomcat keystore file in your PAS for OpenEdge instance with the information from an SSL/TLS certificate obtained from a Certificate Authority (CA).

Note: You will need the password that you used when you ran PKIUTIL to create a public key certificate request file (`MyCert.pk10` in the example procedure described in Obtain a certificate from a Certificate Authority on page 114).

To update a PAS for OpenEdge keystore, perform the following steps:

1. Make a backup copy of the default Tomcat keystore file located in your PAS for OpenEdge *instance-name*`/conf` directory.

   For example:

```
proenv> cd C:\MyInstance\conf
proenv> copy tomcat-keystore.p12 tomcat-keystore.p12.original
```

2. From the *instance-name*`/conf` directory, use the `sslc` command to export the information from the PEM file to the Tomcat keystore.

   Note: The `sslc.exe` executable is an OpenSSL (https://www.openssl.org/) command-line utility that is included in *instance-name*`/bin`.

   For example:

```
proenv> cd C:\MyInstance\conf
proenv> sslc pkcs12 -export C:\Progress\OpenEdge\keys\V.pem -out tomcat-keystore.p12
 -name mysslprivkey
```

   Note: Avoid using upper-case letters in the filename specified after the `-name` parameter. The `sslc` automatically converts the name to lower-case in the alias entry (`Alias name: mysslprivkey`) in the keystore. Using upper-case letters could cause a mismatch when you update the *instance-name*`/conf/catalina.properties` file. See Step 5.

3. When prompted enter the password you used when you created the keystore (i.e. the `.pk1` file). See Obtain a certificate from a Certificate Authority on page 114.
4. Use the `sslc` command to verify what is now in the Tomcat keystore.

   For example:

```
sslc pkcs12 -info -in ./tomcat-keystore.p12
```

5. Update the *instance-name*`/conf/catalina.properties` file to specify the `psc.as.https.keypass` and `psc.as.https.keyalias` properties in the `JSSE keystore` section.

For example:

```
# JSSE keystore used by server.xml for its server key & certificates
psc.as.https.keypass=your_password
psc.as.https.keyalias=mysslprivkey
psc.as.https.storeType=PKCS12
```

Important: The value for `psc.as.https.keyalias` must match the value generated by `sslc` which is always expressed in lower-case letters.

6. Restart the instance and test.

   You can restart the instance using the TCMAN `stop` and `start` actions. Once the instance has restarted, test the connection and authentication configurations from a browser.

Note: You may need to import the CA certificate to the browser.

Configure clients for SSL/TLS

Use the `certutil` command to create the files that supply the necessary keys for connecting to the instance.

To generate the key files and copy them to a instance, perform the following steps:

1. On a PROENV command line, change your working directory to the *OpenEdge-Install-Dir*`\keys\requests`.

   This directory should be where you copied the SSL/TLS certificates as described in Obtain a certificate from a Certificate Authority on page 114.

2. Run `certutil` to import both the public and private SSL/TLS certificates to the *OpenEdge-Install-Dir*`\certs` directory.

   The following is an example with certificates named `MyCertPriv.cer` and `MyCertPub.cer` in the `C:\Progress\OpenEdge\keys\requests` directory:

   ```
   proenv> certutil -import C:\Progress\OpenEdge\keys\requests\MyCertPriv.cer
   Importing trusted certificate to alias name: 9792edd3

   proenv> certutil -import C:\Progress\OpenEdge\keys\requests\MyCertPub.cer
   Importing trusted certificate to alias name: 22c9eb58
   ```

   Note:

   - The `-import` parameter requires the fully qualified pathname of the certificate file.
   - The key file alias names are exactly eight hexadecimal characters in length and have a `.0` (dot-zero) file extension. However, keep in mind that your files will not have the same filenames that are shown in this example.

3. Verify that the key files created in step 2 are located in the *OpenEdge-Install-Dir*`\certs` directory.
4. Copy the key files to the client.

Every client that connects to the instance must have a copy of the key file that was generated by the import.

5. Test the client.

The Spring Security Framework in PAS for OpenEdge

The Progress Application Server (PAS) for OpenEdge employs the Spring Security framework to provide enterprise-level application security with regard to:

- **Authentication** — verification that username and password combinations are valid.
- **Authorization** — enforcing access control (user accounts are granted permissions according to their assigned roles).

Spring Security is a Java-based framework provided in the `/common/lib` directory of the core PAS server. For general information about Spring Security, see http://www.springsource.org. For reference and API information, see http://static.springsource.org/spring-security/site/reference.html

JVM security manager

When you run Tomcat, each instance of the server runs in the context of its own individual JVM whose security can be managed with the Java Security Manger.

PAS is enabled with a default configuration of the standard JVM Security Manager. The JVM security is configured in the `/conf/catalina.policy` file of a given PAS for OpenEdge instance.

The `catalina.policy` file uses Permission classes, either the built-in JDK classes or customized classes, to control what actions a class can take. The following are just a few examples of the built-in classes:

- `java.io.FilePermissions`: controls access to read, write, delete, and other actions for files and directories
- `java.net.SocketPermissions`: controls network access through sockets
- `java.security.SecurityPermissions`: controls access to security methods

For more information on the JVM Security Manager in Tomcat, see http://tomcat.apache.org/tomcat-7.0-doc/security-manager-howto.html

.

Manage users and remote access

PAS for OpenEdge can support one or more Tomcat realm definitions. A *realm* is a collection of usernames and passwords, and it includes the roles associated with each of those users. A *role* controls the amount of access a given group of users has. All authorizations in PAS for OpenEdge are role-based. Access privileges cannot be granted on a user-by-user basis.

PAS for OpenEdge retains the roles defined in the default Tomcat MemoryRealm, but also adds a set of PAS for OpenEdge roles that map to the Tomcat roles and allows for consistency across the product. For example, `ROLE_PSCadmin` allows unrestricted administrator access, and it maps to `admin-gui`, `admin-script`, `manager-gui`, `manager-script`, and `manager-status` roles in Tomcat. These roles are described in Managing users and roles for Manager and Host Manager applications.

Remote access can be handled by using remote access filters, which is described in Secure online deployment of a new ABL application and Remote access filters.

Role-based user authentication

For user authentication, the PAS for OpenEdge comes configured for use with Tomcat MemoryRealm. The MemoryRealm accesses user authentication information stored in *instance-name*`/conf/tomcat-users.xml`. You should replace the MemoryRealm with a different type of realm such as JDBC or LDAP in a production instance. You can have multiple realms of different types configured at the same time.

PAS for OpenEdge retains the roles defined in the default Tomcat MemoryRealm, but PAS for OpenEdge also includes a new set of roles that include the Tomcat roles. The following table lists the new roles and what existing Tomcat roles they map to, if any:

Table 9: PAS for OpenEdge roles mapped to Tomcat roles

PAS for OpenEdge role	Includes Tomcat Manager and Host Manager roles	Description

`ROLE_PSCAdmin`	`admin-gui`
	`admin-script`
	`manager-gui`
	`manager-script`
	`manager-jmx`
	`manager-status`
		Unrestricted administration access.
		• The Tomcat admin roles allow access to the HTML GUI and text interfaces of the Host Manager application
		• The manager roles allow access to the HTML GUI, text interface, JMX proxy, and status pages of the Manager application.

`ROLE_PSCOper`	`admin-script`
	`manager-script`
	`manager-status`
		Restricted operator administration access. The Tomcat roles allow access to the text interfaces of the Host Manager and Manager applications as well as the status pages in the Manager application.

`ROLE_PSCUser`	none	Non-priviliged access
`ROLE_PSCNone`	none	No access, for user account authorization testing

Note: If you use a realm(s) other than the default MemoryRealm, that realm must have the roles listed in the table above.

PAS for OpenEdge comes with three default test accounts configured in *instance-name*`/conf/tomcat-users.xml`, as described in the following table:

Table 10: PAS for OpenEdge default user accounts

Username	Password	Role(s)
tomcat	tomcat	ROLE_PSCAdmin, ROLE_PSCOper, ROLE_PSCUser **Note:** Replace the ROLE_PSCUser in production.
tcuser1	tcuser	ROLE_PSCUser **Note:** Remove the ROLE_PSCUser in production.
tcuser2	tcuser	ROLE_PSCNone **Note:** Remove the ROLE_PSCNone in production.

The format for users entries in *instance-name*/conf/tomcat-users.xml is shown in the following example:

```
<user username="tomcat" password="tomcat"
   roles="ROLE_PSCAdmin,ROLE_PSCOper,ROLE_PSCUser"/>
```

Note: The default username and password should be changed or removed in a production instance.

Remote access filters

The PAS for OpenEdge is configured with filters that allow you to deny access based on IP addresses or host names. These filters are implemented as valves in the /conf/server.xml file of the instance. The following snippet from the default server.xml shows that the filters are set to allow access by anyone:

```
<Valve className="org.apache.catalina.valves.RemoteHostValve"
 allow=".*"/>
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
 allow=".*"/>
```

The RemoteHostValve and RemoteAddrValve filters can take two attributes:

allow

A comma-delimited list of regular expressions that a client must match in order to be allowed access.

deny

A comma-delimited list of regular expressions that, when matched, excludes a client from access.

Note: Enabling remote hostname filters requires DNS hostname resolution, which can result in slower performance.

8

Web application security configurations

In Progress Application Server for OpenEdge, the Spring Security framework implements Web application security.

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 provides the following functionality:

- **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 an OpenEdge ClientPrincipal token that can be used further in the authorization process.

For details, see the following topics:

- Spring Security configuration files

- Enabling security for REST, WEB, APSV and SOAP
- User account configurations
- Login models
- CORS support
- Single sign-on (SSO) support
- Authentication using OpenEdge Authentication Gateway
- Authentication using OAuth2 and JWT
- Authenticate using SAML

Spring Security configuration files

In PAS for OpenEdge, you configure Spring Security by modifying the security properties file (`oeablSecurity.properties`), and the URL access control file `oeablSecurity.csv`) instead of changing bean constructor and property settings in multiple Spring Security XML files. These external properties files are designed to be release independent, to be open to external management, and to provide access to all of the security process functionality in the Spring Security framework.

Note: The Spring Security XML files are in the `WEB-INF/spring` directory on the server, but there is usually no need to edit them directly.

Security properties files

The `oeablSecurity.properties` file can exist in three-tiered hierarchy so that you can:

1. Configure properties that are used for all ABL applications deployed on the instance by setting properties in *`instance-name`*`/conf/oeablSecurity.properties`. The defaults established in this file can be overridden by properties set in an ABL application (#2) or a web application (#3).
2. Configure properties that are used for ABL applications by setting properties in *`instance-name`*`/ablapps/`*`abl-app-name`*`/oeablSecurity.properties`. Property settings in this file override the server instance defaults (#1).

   Note: This level is optional. It is useful when you have more than one ABL application deployed on an instance and when those ABL applications require different security configurations.

3. Configure properties for a specific web application in an ABL application by setting properties in *`instance-name`*`/webapps/`*`web-app-name`*`/WEB-INF/oeablSecurity.properties`. Property settings in this file override the server instance defaults (#1) and the ABL application defaults (#2).

The Spring Security properties and their values are documented in the `oeablSecurity.properties.README` file.

For some properties, the oeablSecurity.properties file employs a naming convention (*bean-name*.*property*=*value*) that simplifies the identification of a bean and its properties. For example:

```
OEClientPrincipalFilter.sealAnonymous=false
```

Note:

Beginning with the OpenEdge 11.7 release, *property*=*value* pairs replace references to specific XML security model configuration files. For example, if you want to specify basic HTTP authentication using a local users file you would specify this set of properties in oeablSecurity.properties:

```
http.all.authmanager=local
client.login.model=basic
```

Prior to the OpenEdge 11.7 release, you would have to have to sort through a number XML templates to find one that matches the required security model, update it, and reference it from the web.xml file.

URL access control file

The *instance-name*/webapps/*web-app-name*/WEB-INF/oeablSecurity.csv file implements URL access controls for web applications. Each entry (or line) in the file is and ordered set of three values.

Note: Access control lists, since they are ordered sets of three values, do not lend themselves well to the format of a properties file with its name/value pairs. Therefore, URL access controls were not included in the oeablSecurity.properties file. CSV files are more suitable for access control lists, and they are easily maintainable by many external administrative tools.

The three values of an entry in oeablSecurity.csv correspond to the three attributes of a Spring Security intercept-url element, namely:

- *pattern* — the URL pattern which can include wildcards and regular expressions
- *method* — the HTTP access method
- *access* — role[s] that are allowed access to the resource

Each line in the file must contain all three values specified in a comma separated list.

For example, the following snippet grants access to any user who has either ROLE_PSCAdmin or ROLE_PSCUser privileges to data from a resource whose URL begins with /web/sales/.

```
"/web/sales/**", "GET", "hasAnyRole('ROLE_PSCAdmin','ROLE_PSCUser')"
```

Enabling security for REST, WEB, APSV and SOAP

Spring security is automatically enabled for validating clients of REST and WEB applications. You must update `oeablSecurity.properties` to enable Spring Security to validate clients of APSV and SOAP applications.

For APSV, set the `apsv.security.enable` property to `basic`. By default, the property is set to `none`.

For SOAP, set the soap.security.enable property to `basic`. By default, the property is set to `none`.

For more information, see the `oeablSecurity.properties.README` file

User account configurations

You configure user account sources for web applications by setting the `http.all.authmanager` property in the `oeablSecurity.properties` file. The syntax is:

```
http.all.authmanager={ local | extlocal | ldap | ad | oerealm }
```

For more information see the `oeablSecurity.properties.README` file.

See also

- Local and Extended Local user accounts on page 126
- LDAP user accounts on page 128
-
- OERealm user account configurations on page 129

Local and Extended Local user accounts

Both local and extended local configurations use an ABL application's `WEB-INF/users.properties` file as the source for user account information.

Local configurations

The local model is usually used during application development, where you would want a quick way of testing a web application's security structure. It allows you to easily edit accounts and add clear-text passwords in order to test user access based on roles.

To implement the local model, add the following setting in the *Security and authentication models* section of the `oeablSecurity.properties` file:

```
http.all.authmanager=local
```

See the `oeablSecurity.properties.README` file for more information.

After choosing the local security configuration model for a web application, you can add, remove, modify users or user authentication settings by updating the ABL application's `WEB-INF/users.properties` file.

Extended local configurations

Like the local model, the extended local model uses the `WEB-INF/users.properties` as the source for user account information . However, the extended local model is more secure because it allows you to use encrypted passwords. Encrypted (i.e. hashed/salted) passwords are generated with the GENSPRINGPWD utility. See Generate encrypted passwords with GENSPRINGPWD on page 127

To implement the extended local model, add the following setting in the *Security and authentication models* section of the `oeablSecurity.properties` file:

```
http.all.authmanager=extlocal
```

See the `oeablSecurity.properties.README` file for more information.

The users.properties file

The following shows the syntax of entries in `users.properties`:

```
username=password, role1, [role2, . . ., roleN, ] { enabled | disabled }
```

The following shows the content of the default `users.properties` file:

```
restuser=password,ROLE_PSCUser,enabled
restdebug=password,ROLE_PSCUser,ROLE_PSCDebug,enabled
```

The contents of the `users.properties` file are included as an example and they are not secure since they are stored as plain text. Typically, for a production server, you would generate encrypted passwords, and add the secure passwords to the `users.properties` file. See Generate encrypted passwords with GENSPRINGPWD on page 127 for more information.

Generate encrypted passwords with GENSPRINGPWD

To provide greater security for user accounts stored in `users.properties`, you can use the GENSPRINGPWD command-line utility (*oe-install-dir*`/bin/genspringpwd`). The utility accepts a clear-text password as input and generates an encrypted password. `bcrypt` is the default and only supported hashing algorithm.

GENSPRINGPWD Syntax

```
genspringpwd password
```

password

A character string representing the password to be encrypted.

Store encrypted passwords

After you have created an encrypted password with GENSPRINGPWD, copy and paste the generated value to the *password* field in the corresponding `users.properties` entry. Use the following syntax to store passwords in `users.properties`:

```
username=password, role1, [role2, . . ., roleN, ] { enabled | disabled }
```

LDAP user accounts

The Lightweight Directory Access Protocol (LDAP) is a protocol for authenticating user credentials that are stored on an LDAP-compliant server. If you are not familiar with LDAP implementation and user account management, refer to LDAP administration documentation online.

To implement the local model, add the following setting in the *Security and authentication models* section of the `oeablSecurity.properties` file:

```
http.all.authmanager=ldap
```

You configure LDAP in the *LDAP Authentication Manager* section of the `oeablSecurity.properties` file. See `oeablSecurity.properties.README` for an explanation of LDAP properties and their valid settings.

Note: The LDAP UserDetails implementation in Spring Security automatically updates role names. Ensure that role names are upper case since searches are case sensitive.

After configuring the LDAP model for your Web application, you can implement CORS support.

Active Directory user accounts

MicroSoft's Active Directory (AD) service provides a namespace for resolving wide range of objects including user accounts. An Active Directory configuration for web applications deployed on PAS for OpenEdge allows authentication using a Windows `user@domain` type ID. It locates and validates users from the user object's *userPrincipalName* field. It obtains granted roles from the user object's *memberOf* field.

To implement the Active Directory security model, add the following setting in the *Security and authentication models* section of the `oeablSecurity.properties` file:

```
http.all.authmanager=ad
```

You configure AD in the *Microsoft Active Directory user account services* section of the `oeablSecurity.properties` file. See `oeablSecurity.properties.README` for an explanation of AD properties and their valid settings.

Note: If more configuration detail is needed, use the full LDAP configuration described in the *LDAP Authentication Manager* section of the `oeablSecurity.properties.README` file. .

OERealm user account configurations

The OERealm model allows you to use an OpenEdge application server as a source of user account information for the authentication process.

To implement the OERealm security model, add the following setting in the *OERealm user account server configuration* section of the `oeablSecurity.properties` file:

```
http.all.authmanager=oerealm
```

You configure it in the *OERealm user account server configuration* section of the `oeablSecurity.properties` file. See `oeablSecurity.properties.README`for more information.

Note that an OERealm implementation consists of the following components:

- **OpenEdge Realm (OERealm) Client:** The client contains the Spring Security authentication provider and user account service objects that interact with the application server 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 application server.
  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:**An ABL class that runs on the AppServer and responds to requests from the OERealm client and implements the interface IHybridRealm.

  The actions of the class is 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 more information

Note: For more information about OERealm, see *OpenEdge Application Server: Administration*.

Creating an OERealm service interface

To implement OpenEdge SPA, you must create an ABL class that implements the OERealm service interface that finds user account information, obtains user attributes, and validates a user account's password.

To configure an OERealm service:

1. Configure a OpenEdge Application Server to host the OERealm service interface class.

   Note: Either a PAS for OpenEdge or an OpenEdge AppServer instance can host the OERealm service interface class. If you are using the OpenEdge AppServer, it must be configured as a STATE-FREE instance. (A PAS for OpenEdge instance functions in any operating mode.)

2. Create a singleton OOABL class that implements the OERealm HybridRealm 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 11: Methods defined in the HybridRealm.cls class

Method	Mandatory?	Description
`GetAttribute`	Yes	This method retrieves an attribute for the specified user account.
`GetAttributeNames`	No	This method retrieves the name assigned to the attributes.
`GetUserNames`	No	This method retrieves all the user names associated with the user account.
`GetUserNamesByQuery`	No	This method retrieves the user names that match the input query string.
`RemoveAttribute`	No	This method deletes the value of the given attribute.
`SetAttribute`	No	This method assigns a new value to the given attribute.

Method	Mandatory?	Description
ValidatePassword	Yes	This method validates the client-supplied password of the user account against the stored password. **Note:** You can define both the clear-text and digest form of the ValidatePassword() method.
ValidateUser	Yes	This method searches for the user account, validates whether the account domain is valid and enabled, and returns the account numeric ID. **Note:** 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.

You can implement the GetAttribute() method to return the UNKNOWN or CHARACTER value. If you have configured the Web application 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 12: OERealmUserDetailsImpl module attributes

Attribute	Data type	Character encoding format
ATTR_ROLES	string-extent	"" or "xxxx[,yyyy,zzzz]" **Note:** The OERealm service interface must return roles without the ROLE_ prefix. For example, the interface must return PSCUser instead of ROLE_PSCUser. These roles are used in oeablSecurity-xxxxx-oerealm.xml grant/deny access to URL/method combinations and will be used by the external security as part of its authorization functions.
ATTR_ENABLED	boolean	A character from the {T,t,F,f,Y,y,N,n,0,1} set. T,t,Y,y,1 denotes that the user is enabled. F,f,N,n,0 denotes that the user is disabled.

Attribute	Data type	Character encoding format
ATTR_LOCKED	boolean	A character from the {T,t,F,f,Y,y,N,n,0,1} set. T,t,Y,y,1 denotes that the user is locked. F,f,N,n,0 denotes that the user is not locked.
ATTR_EXPIRED	boolean	A character from the {T,t,F,f,Y,y,N,n,0,1} set. T,t,Y,y,1 denotes that the user is expired. F,f,N,n,0 denotes that the user is not expired. Spring Security does not allow expired users.

3. Build and test your OERealm service interface. You can perform this step using an ABL client test harness that calls the OERealm service interface.

Note: Errors returned to the client are generic. You can find error details that are more helpful in the application's log file.

OERealm security considerations

The OERealm authentication process must be secure against unwarranted attacks.

The following considerations must be observed for security purposes:

- **OERealm service interface security**

  If the ABL class that provides the OERealm service requires authentication of clients, then the client must have a realm token property configured with a file that contains a sealed ClientPrincipal that the OERealm service is capable of validating.

  Since the OERealm user details service can be configured to send a sealed ClientPrincipal to the AppServer OERealm service interface on each method it invokes, 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 ("oech1::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 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 properties.**

You can enhance security by setting the following OERealmAuthProvider properties in the`oeablSecurity.properties` file while sealing the ClientPrincipal:

- The `OERealmAuthProvider.userDomain` attribute to set the static OpenEdge domain.
- The `OEClientPrincipalFilter.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 Web application just by disabling the domain. The following table defines the available combinations of domain, domain access code, and user ID:

Table 13: Available combinations of domain, domain access code, and user ID

Domain name	Key attribute	User ID	Signing operation
"" (default)	"" (default)	"uid" ("uid@")	Seal the blank domain using the built-in Domain Registry blank access code.
""	"<xxxx>"	"uid" ("uid@")	Seal the blank domain using the clear text sting value, "<xxxx>".
""	"oech1::<xxxx>"	"uid" ("uid@")	Seal the blank domain using the encoded value, <xxxx>.
"abc"	"" (default)	"uid@abc"	Seal the abc domain using the built-in Domain Registry blank access code.
"abc"	"<xxxx>"	"uid@abc"	Seal the abc domain using the clear text sting value, "<xxxx>".
"abc"	"oech1::<xxxx>"	"uid@abc"	Seal the abc domain using the encoded value, <xxxx>.

- **Set the multiTenant property.**

  If you enable the `OERealmAuthProvider.multiTenant` property , all the user IDs are appended with the `userDomain` value only if the user-id being authenticated does not already specify a Domain name..

- **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. These values are blank by default.
  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 the Web application's `OERealmAuthProvider` supersedes 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 the `OEClientPrincipalFilter.enabled` 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 OpenEdge application server . To use this validation technique, the Web application's `OERealmUserDetailsImp` module acts as a client and identifies itself to the application server'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 Web application's `OERealmUserDetails` for acting as an OpenEdge application server client:

1. Use the `genspacp` utility, at the `$DLC/bin` location, to create a sealed ClientPrincipal that represents a OERealm service client. The utility requires the user ID, domain name, domain access code, and output file name to create a binary file that contains the sealed ClientPrincipal in the `<file-name>.cp` format. The domain name and domain access code correspond to those that are then used in the OERealm service interface to validate clients.
2. Copy the output file from the `$DLC/bin` location to the Web application's `WEB-INF/classes` directory.
3. Set the `OERealm.UserDetails.realmTokenFile` property 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 Web application is loaded, if the `OERealm.UserDetails.realmTokenFile` property 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.

Multi-domain support for user accounts

This topic describes how PAS for OpenEdge extends multi-domain support for user accounts.

Simple multi-domain support in PAS for OpenEdge accommodates implementations where the user-id entered by the client is a fully qualified OpenEdge ID. *Fully qualified* means that both the user-id and the domain (or tenant) name are included. This type of user authentication requires the back-end user account storage to be implemented with multiple user domain capabilities.

Note:

An OpenEdge *domain* is a group of user accounts that all share the same access control rights. A *tenancy* is a security feature that attributes access control to data for all user accounts of all the domains associated with a single tenant. A *tenant* is required to have one domain in which all user account members are members of the tenant. Any tenant may be associated with multiple domains.

In addition, PAS for OpenEdge includes multi-domain support for implementations where the back-end system's user account storage does not specify domains or tenancy. The primary example of this is an LDAP authentication provider. The user account storage on an LDAP server is a flat space holding all user accounts for all domains, and each account has a unique ID. In this case, the domain name is not something the client supplies during authentication, but rather something that must be related to the user account by configuring the OEClientPrincipalFilter bean in the *OEClientPrincipalFilter bean* section of the `oeablSecurity.properties` file. It has capabilities that allow it to derive a domain name from a user account's granted roles.

This method of using Spring Security roles as OpenEdge domains follows the logical pathway where a user account becomes a member of an OpenEdge domain by virtue of being granted membership in the domain. This is the same reasoning used in LDAP directories when an administrator associates individual user accounts with an LDAP group for the purpose of controlling access.

For more information on multi-domain support for authentication systems like LDAP see the following topics:

- Resolving user-id and domain names on page 135
- Obtaining a domain name from a role name on page 136
- Configuring multiple domains on page 136
- OpenEdge domains in LDAP configurations on page 138

Resolving user-id and domain names

The OEClientPrincipalFilter contains the logic that processes an OpenEdge ID to yield a user-id and domain name. The user-id and domain are used in creating a client-principal object that is subsequently passed to the ABL application code.

The OEClientPrincipalFilter logic follows these general steps after an authentication provider has successfully authenticated the user:

1. If the input OpenEdge ID is fully qualified (user-id plus domain name), proceed to Step 3.
2. Try to resolve the domain name by doing the following:
   a. First, look for a user role in the `OEClientPrincipalFilter.domainRoleFilter` property, which is recognized as holding a domain name. If the domain name is found, proceed to Step 3.

      Note: All OEClientPrincipalFilter properties are set in the `oeablSecurity.properties` file.

   b. Next, try using the `OEClientPrincipalFilter.domain` property value. If the domain name is found, proceed to Step 3.
   c. If the domain name remains unresolved after Steps 2a and 2b, raise an error condition.
3. Create a client-principal object and populate its data fields.
4. If the `OEClientPrincipalFilter.key` property is configured, use its value as the Domain Access Code (DAC) for all domain names.

Note: The `OEClientPrincipalFilter.key` and the`OEClientPrincipalFilter.registryFile` properties are mutually exclusive. Both cannot be configured at the same time.

5. If the `OEClientPrincipalFilter.registryFile` property is configured, use the domain name (either input directly in Step 1 or resolved by one of the methods in Step 2) to look up the domain's DAC.
6. If the DAC cannot be resolved, raise an error condition.
7. Seal the client-principal with the DAC and store it in the user's HTTP login session context for use in subsequent client requests.

Obtaining a domain name from a role name

The `OEClientPrincipalFilter.domainRoleFilter` property of the Spring Security OEClientPrincipalFilter bean obtains an OpenEdge domain name from a user-granted role. The `domainRoleFilter` property is a Java RegEx (regular expression) pattern that is used to identify role names produced by Spring Security and to extract the domain name from them. The basic requirement is to adopt a role-naming convention that distinguishes role names that represent domains from other role names.

The Java RegEx expression can be any valid pattern with a single extraction group (a Java RegEx pattern enclosed in parentheses). The full pattern is used to match a role name. Anything that matches the pattern enclosed in parentheses is extracted and becomes a domain name.

The following are examples of user role-naming conventions that define domains:

`OED:(.*)`	The domain name is everything after `OED:`
`Domain-(.*)-name`	The domain name is all characters between `Domain-` and `-name`

Spring Security authentication providers (the beans that authenticate user accounts) prefix `ROLE_` to all role names provided for authenticated users. Since the role names presented to the OEClientPrincipalFilter bean are in the form `ROLE_`*`user-role-name`*, this is what would be matched by the `domainRoleFilter` property:

- `ROLE_OED:(.*)`
- `ROLE_Domain-(.*)-name`

When the `domainRoleFilter` property is not defined or is a blank string, looking for a domain name in the user's list of granted roles is disabled.

Configuring multiple domains

Configuring for multi-domain support begins with the follow steps:

1. Design the OpenEdge domain name space to be used by the ABL application, and assign a Domain Access Code (DAC) to each domain name.
2. Build a CSV text file with domain names and clear-text DAC pairs, one pair per line. Domain names and DACs are separated by a comma. For example:

```
domain-name,clear-text-DAC
```

3. Use the gendomreg utility to create an encrypted Java keystore file from the CSV file. For example:

```
oe-install-dir/bin/gendomreg domreg.csv domreg.keystore
```

4. Copy the encrypted Java keystore file to each PAS for OpenEdge instance's `.../conf` directory.
5. Configure the OEClientPrincipalFilter bean in the `oeablSecurity.properties` file as follows:
   a. Set the `OEClientPrincipalFilter.key` property to a blank value.
   b. Set the `OEClientPrincipalFilter.registryFile` property and set its value to the file name of the Java keystore that you created in Step 3. For example:

```
OEClientPrincipalFilter.registryFile=domreg.keystore
```

   c. Set the `OEClientPrincipalFilter.domain` property to a default domain that has minimal access rights to your ABL application.
   d. Optionally configure `authz`, `accntinfo`, `roles`, and other properties.

The next step is to design the role name pattern that will be used to designate an OpenEdge domain. Ideally the name pattern will have a unique prefix that distinguishes it from all other names. The name pattern may also employ a postfix to help establish uniqueness. See Obtaining a domain name from a role name on page 136 for more information about designing a role name pattern.

Note: Since the OEClientPrincipalFilter bean uses Java RegEx functionality, create a role name pattern that is easy for Java RegEx to parse.

When you have established a role name pattern, construct the Java RegEx pattern that will distinguish the domain name from any other normal user account granted roles. For example, with `OED:`*`domain-name`* as the domain name:

Table 14: Spring users.properties accounts

Role Name	Spring Security Role Name	Java RegEx Match Pattern
ROLE_OED:acme	ROLE_OED:acme	ROLE_OED:(.*)

Table 15: Spring LDAP groupOfUniqueNames object

Role Name	Spring Security Role Name	Java RegEx Match Pattern
cn=OED:acme	ROLE_OED:ACME	ROLE_OED:(.*)

The Spring LDAP Authentication Provider prefixes `ROLE_` to any LDAP group name it uses as a user account role, and it converts the LDAP group name to upper case.

Finally, set the OEClientPrincipalFilter bean's domainRoleFilter property to the value of the Java RegEx match pattern.

Note: If your Java RegEx match pattern contains a back slash (\ ), you may be required to escape it with another backslash (\\ ).

OpenEdge domains in LDAP configurations

Spring Security LDAP configuration details will almost always be unique to each individual enterprise; however, the following will give you information about configuring OpenEdge domains in the directory service so that the OEClientPrincipalFilter bean can find and use them.

Each LDAP directory service has a schema that defines its objects and object attributes. Different implementations of a directory service may vary in the objects and object attributes it supports. Examples of different directory services are:

- Apache Directory Service
- Windows Active Directory
- OpenLDAP

An OpenEdge domain will be just another LDAP group object that contains a list of the user accounts (Distinguished Names) who are members of that group, and who are entitled to the access associated with the group. The difference is that the LDAP group's name will be based on the name pattern you defined in the previous sections.

For example, directory service administrators use the LDAP `groupofUniqueNames` objects as user account roles, where each group member's user account is in a `uniqueMember` attribute. The directory service's hierarchical object space might look like the following:

```
dc=acme
   ou=users
      cn=acmeuser1,ou=users,dc=acme,dc=com
   ou=groups
      cn=OED:acme,ou=groups,dc=acme,dc=com
      uniqueMember=cn=acmeuser1,ou=users,dc=acme,dc=com
```

In this simple example, the user account `acmeuser1` is in the domain `acme`. This occurs because a group is defined with the name `OED:acme`, and the user account for `acmeuser1` is a member of that group. Therefore, when Spring Security authenticates `acmeuser1`, it will see that `acmeuser1` is a member in the `ROLE_OED:acme`. The role supplies the domain name `acme` for creating a client-principal's `domain-name` field.

For more information about LDAP, see LDAP user accounts on page 128

Login models

PAS for OpenEdge supports the following login models for web applications:

- `basic` — typically used by client authentication to stateless ABL business application architectures. There are no configurable properties.
- `anonymous` — used as the default for getting started with the development of an ABL web application. It allows unauthenticated client access to the ABL web application until it is mature enough to use product quality authentication. There are no configurable properties.

- `container` — used when the ABL web application wants to use the Realm authentication provided by the Apache Tomcat web server it runs in. This can be useful when synchronizing user accounts with other 3rd party Java web applications in the same PAS for OpenEdge instance. Set the http.jee.all.mappableRoles property to a comma separated list of role names from the Tomcat realm's authentication.
- `form` — typically used for managing the login/logout of clients to a stateful ABL business application architecture. Properties are found in the *Form login Filter bean* and *Logout Filter bean* section of the `oeablSecurity.properties` file.
- `sso` — login via a ClientPrincipal token that represents a user. SSO properties are found in the *OpenEdge SSO support* section of `oeablSecurity.properties`. Also see Single sign-on (SSO) support on page 140.

You set the login model as a value of the `client.login.model` property. See `oeablSecurity.properties.README` for more information.

CORS support

Cross-origin resource sharing (CORS) is a W3C group standard that allows an HTTP client to access a web page from one domain in order to access resources located in another domain. Such "cross-domain" requests are otherwise forbidden by a default, same-origin security policy of a web browser. The CORS standard defines a way in which a client can ask a web server if it can access the cross-origin request. The configuration of the web server determines if the cross-domain request will be granted.

The CORS standard was designed primarily to support JavaScript clients, however it may be applicable to other clients as well.

The CORS standard is implemented through HTTP headers that allow resource access to permitted domains. Most modern browsers support these headers and enforce the restrictions these headers establish. Additionally, for HTTP request methods that might affect user data (in particular, methods other than `GET` or `POST` with certain MIME types), the specification mandates that the browser preflight the request. Preflighting is a check to determine if the HTTP request is safe to send to the other domain. The server solicits supported methods from the other domain with an `HTTP OPTIONS` request header. Upon approval of the preflight request, the actual HTTP request is sent. Servers can also notify clients whether user credentials (including cookies and HTTP authentication data) can be sent with requests.

Note:

Although CORS support is extended to modern browsers, check to ensure that your browser supports the CORS standard.

You must manage security constraints (for example, roles) in the `oeablSecurity.properties` file of the web application. See the *OECORSFilter bean* of the `oeablSecurity.properties.README` for more information.

A CORS enabled server or web application classifies all HTTP requests as:

- A CORS request that contains the 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

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 web application security policy, and is enabled for all generic requests and CORS requests from any domain.

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 the 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 the XMLHttpRequest API, 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.

Use CORS

To add CORS support with a PAS for OpenEdge instance, perform the following steps

1. Open the `WEB-INF/oeablSecurity.properties` file of your web application.
2. Find the `OECORSFilter` bean.
3. Set `OECORSFilter` properties.

   See the `WEB-INF/oeablSecurity.properties.README` for information about OECORSFilter properties.
4. After saving your changes, restart the PAS for OpenEdge instance.

Single sign-on (SSO) support

Single sign-on (SSO) in OpenEdge refers to the creation of a sealed security token called the `ClientPrincipal` that represents a user identity within a security domain. The `ClientPrincipal` is created after Spring Security successfully completes authentication and authorization. The `ClientPrincipal` retains the user credentials and remains available within a web application, eliminating the need for a user to login again.

In PAS for OpenEdge, basic Spring Security token generation is extended to create an OpenEdge `ClientPrincipal` object. Creation of the `ClientPrincipal` is enabled by default by this property in the `oeablSecurity.properties` file of the web application:

```
OEClientPrincipalFilter.enabled=true
```

Set the property to `false` to disable the `ClientPrincipal`. You can set other `ClientPrincipal` properties in the `OEClientPrincipalFilter` bean section of the `oeablSecurity.properties` file. In particular, you will want to set the `OEClientPrincipalFilter.domain` property to specify where the `ClientPrincipal` is valid.

After domain and other properties are set for the `ClientPrincipal`, you can configure SSO properties in the `OpenEdge SSO support` section of the `oeablSecurity.properties` file.

See the `oeablSecurity.properties.README` file for more information on the properties and values for the `ClientPrincipal` and for SSO.

Note: In PAS for OpenEdge, SSO is available for client access via the APSV and REST transports but not for the SOAP transport.

Authentication using OpenEdge Authentication Gateway

OpenEdge Release 11.7.2 extends the functionality of the OpenEdge Authentication Gateway to be a Security Token Service (STS) for OpenEdge applications hosted on a PAS for OpenEdge instance. In prior releases, its use was limited to instances and database connections.

The Authentication Gateway is a secured HTTPS server that provides STS functionality, which includes user authentication and Client-Principal generation. The use of a common Authentication Gateway for instances, databases, and applications enhances security and reduces complexity (because, for example, each PAS for OpenEdge server instance does not need to maintain its own copy of OpenEdge Domains and Domain Access-codes).

Connections between a PAS for OpenEdge instance and an Authentication Gateway server are facilitated through the STS AuthenticationProvider, a plug-in to the Progress Application Server's Spring Security framework.

For more information about the basic architecture of the STS AuthenticationProvider and its configuration, see the following topics:

- The STS AuthenticationProvider on page 141
- Configuration and Testing on page 142

The STS AuthenticationProvider

The STS AuthenticationProvider in PAS for OpenEdge is a plug-in that enables the use of the OpenEdge Authentication Gateway as the source for user authentication and client-principal token generation.

The STS AuthenticationProvider:

- Takes user credentials (i.e. user name and password) as input
- Connects to the Authentication Gateway via a secure HTTPS connection
- Sends user credentials to the Authentication Gateway for authentication

If the authentication is successful, the STS AuthenticationProvider:

- Receives a sealed client-principal token from the Authentication Gateway
- Passes the client-principal (unaltered) to the Spring Security URL authorization process

Spring Security URL authorization:

- Performs Role-Based Authorization (RBA)
- Passes the client-principal token (if RBA is successful) to ABL applications

Note: The client-principal's `ROLES` attribute serves as input to the Spring Security URL RBA process. Therefore, the `ROLES` attribute must contain at least one role name that meets Spring Security's URL access control requirements.

Like other Spring Security components in PAS for OpenEdge, the STS AuthenticationProvider is configured in an `oeablSecurty.properties` file on the instance, ABL application, or web application levels. (For more information about the hierarchy of these `oeablSecurty.properties` files, see the *Security properties files* topic in *Progress Application Server for OpenEdge: Administration Guide*.)

On the instance level (`../conf/oeablSecurity.properties`, the file contains a complete set of STS AuthenticationProvider properties. For example:

```
## OpenEdge Authentication Gateway client configuration
 ## for direct user logins to a PASOE server
 ##
 sts.AuthProvider.multiTenant=true
 sts.AuthProvider.userDomain=
 sts.UserDetails.stsURL=https://host:port
 sts.UserDetails.stsKeystore=
 sts.UserDetails.clientHeaderName=x-oests-token
 sts.UserDetails.noHostVerify=false
 sts.UserDetails.certLocation=${psc.as.oe.dlc}/certs
 sts.UserDetails.tlsCipherSuites=
 sts.UserDetails.tlsProtocols=
 sts.UserDetails.userAgent=PASOE (Spring)
 sts.UserDetails.sniHost=
```

The files on the ABL application (`../ablapps/`*`abl-app-name`*`/oeablSecurity.properties`) and the web application levels (`../webapps/`*`web-app-name`*`/WEB-INF/oeablSecurity.properties`) contain the properties that can override the properties specified on the instance level. For example:

```
## OpenEdge Authentication Gateway client configuration
 ## for direct user logins to a PASOE server
 ##
 sts.AuthProvider.multiTenant=true
 sts.AuthProvider.userDomain=
 sts.UserDetails.stsURL=https://host:port
 sts.UserDetails.stsKeystore=
 sts.UserDetails.noHostVerify=true
 sts.UserDetails.sniHost=
```

See the `../conf/oeablSecurity.properties.README` file for a description of the STS AuthenticationProvider properties and an explanation of their valid values.

Configuration and Testing

To use the OpenEdge Authentication Gateway as an STS authentication provider for a PAS for OpenEdge instance, you must configure the instance as an Authentication Gateway client. The following sections describe how to setup, configure, and troubleshoot an Authentication Gateway client:

- OpenEdge Authentication Gateway configuration on page 143
- Configuring a web application on page 144
- Troubleshooting the PAS for OpenEdge instance on page 145

See *OpenEdge Authentication Gateway Utilities* in the *OpenEdge Gateway Guide* for information about the command line tools that are available to configure and manage the Authentication Gateway.

OpenEdge Authentication Gateway configuration

The OpenEdge Authentication Gateway must have a valid HTTPS TLS (Transport Layer Security, the successor to SSL) certificate and client-key installed.

If it hasn't been done already, an administrator can perform the installation using the OpenEdge STS Key Utility. For example:

```
oe_install_dir/bin/stskeyutil install -url https://oeag-dns-name:oeag-port-number
 -file oeagserverkey.ecp
```

Important: Always remove the `.ecp` file from the server after an install to prevent unauthorized access to your configuration. Store it in a secure location in case you need it to re-configure at a later date.

Test the installation with the STS Client Utility. For example:

```
oe_install_dir\bin\stsclientutil -url https://yourmachinename:port -cmd ping
       [-nohostverify]
```

Note:

Be aware of these differences between a development server (used for application development) and a production server (used for application deployment):

- For development servers, the Authentication Gateway and its clients ship with a server certificate specifically crafted to allow TLS connections, but which will fail post-connection Hostname validation. Therefore, the client connections to an Authentication Gateway always must include the `-nohostverify` option to work. For example:

  The `-nohostverify` option allows full server certificate validation per Public Key Cryptography Standards (PKCS) standards, but skips post-connection Hostname validation.

- For production servers, the OEAG server must obtain, configure, and use a non self-signed server certificate. Therefore, it is required that the ROOT CA and any optional Intermediate CA certificates needed to validate the OEAG server's certificate be installed into the OpenEdge installation's DLC/certs directory. Install the ROOT CA and Intermediate CA certificates using only the OpenEdge supplied DLC/bin/certutil utility. Any other installation mechanism is not guaranteed to work.

After verifying that the Authentication Gateway is running, test HTTP connectivity:

```
oe_install_dir/bin/stsclientutil -cmd ping -url
https://oeagd-dns-name:oeag-port-number  [ -nohostverify ]
```

See *OpenEdge Getting Started: OpenEdge Authentication Gateway Guide* for more information about client-key generation, installation, and testing.

Testing the connection

Once the HTTPS (TLS) certificates and client-key are installed, they should be tested to ensure that the PAS for OpenEdge instance can connect to the OpenEdge Authentication Gateway. To test the connection:

1. Test access to the installed Authentication Gateway client-key with the `stskeyutil verify`command. For example:

```
oe_install_dir/bin/stskeyutil verify -file oeagserverkey.ecp -info
```

   Successful verification is indicated by the following message:

```
Server key validation success
```

2. After verifying that the Authentication Gateway is running, test HTTP connectivity to the OEAG server (ensure the server is running before you do this)

```
oe_install_dir/bin/stsclientutil -cmd ping -url
https://oeagd-dns-name:oeag-port-number  [ -nohostverify ]
```

Note: Only use `-nohostverify` when connecting to an Authentication Gateway that is using the OpenEdge-supplied default server certificate, which is used in application development or testing situations but not for actual application deployment.

See STS Key Utility (stskeyutil) on page 151 and STS Client Utility (stsclientutil) on page 148 for more information about these utilities and their command-line options.

Configuring a web application

To configure a web application to use the OpenEdge Authentication Gateway, set the following properties in the PAS for OpenEdge web application's *instance_name*`/webapps/`*Web_app_name*`/WEB-INF/oeablSecurity.properties` file:

1. `http.all.authmanager=sts`

   Enables the use of an STS AuthenticationProvider.

2. `sts.UserDetails.stsURL=https://`*oeag-dns-name*`:`*oeag-port-number*

   Specifies the URL for the Authentication Gateway connection.

3. (Optional) `sts.UserDetails.noHostVerify=true`

   Enable the `-nohostverify` option to turn off host verification. (Only for servers used for application development and testing.)

4. (Optional) `sts.AuthProvider.userDomain=`*sts-configured-domain-name*

   Specify a fixed OpenEdge domain name that is appended to the client's user-id before authenticating with the Authentication Gateway's STS.

5. (Optional) `sts.UserDetails.stsKeystore=`*`client-key_pathname`*

   Specify the directory where the STS AuthenticationProvider looks for the Authentication Gateway's client key file. The STS AuthenticationProvider searches `oeablSecurity.properties` files for a specified `sts.UserDetails.stsKeystore` value in the following order and uses the first value it finds:

   a. `$CATALINA_BASE/webapps/<web-app-name>/WEB-INF/`

   b. `$CATALINA_BASE/ablapps/abl-app-name/conf/`

   c. `$CATALINA_BASE/conf/`

   d. `$CATALINA_HOME/conf/`

   If the path to the client key file is not specified in any of those `oeablSecurity.properties` files, the value of the `STSKEYSTORE` multi-session Agent process environment variable is used. And if `STSKEYSTORE` is not set, the default client key file in *`openedge_install_dir`*`/keys` is used.

6. (Optional) `sts.UserDetails.sniHost=`*`virtual-hostname`*

   Use this property when the OpenEdge Authentication Gateway server is configured with multiple virtual hosts that are bound to a single IP address. Specify the hostname that you want your web application to connect to. The web application requests the virtual host's SSL certificate during the SSL handshake instead of the Server URL's host.

Next, edit the web application's URL access control file, *`instance_name`*`/webapps/`*`Web_app_name`*`/WEB-INF/oeablSecurity.csv`. If your URL access controls are role-based (I.e.when a user must be assigned a particular role to gain access), you must change the `hasRole(...)` field to include one of the role names inserted into a client principal issued by the Authentication Gateway's STS.

Note: If the web application's URL access controls use `hasRole(...)` and the client principal issued by the Authentication Gateway's STS does not include a role attribute, access to some of the application's URLs will be rejected

---

Finally, optionally specify any advanced STS AuthenticationProvider properties, which include, but are not limited to:

- Customized TLC connection attributes
- A customized HTTP header name that is used to pass client-key authorization to an STS
- A customized HTTP User-agent header value that identifies the client to an STS

Detailed description of these advanced properties can be found in the *`instance_name`*`/conf/oeablSecurity.properties.README` file.

Troubleshooting the PAS for OpenEdge instance

After changing any Spring Security configuration in a PAS for OpenEdge instance, you should restart the instance and ensure that the changes are of the right data type, value, and definition. When the server completes its startup operations, check for errors in all of the log files, and determine if they are related to the Spring Security configuration changes. The Spring Security configuration errors are normally found in the instances's *`instance_name`*`/logs/localhost.`*`date`* log file. The error will always contain an extensive message indicating that Spring could not create a bean, and will be followed by a Java exception stack dump. The last part of the message will indicate which Spring bean failed to start and what the reason is.

Once the PAS for OpenEdge instance starts without errors, test user authentication and URL access by logging in to the web application .

If user authentication fails, investigate other Authentication Gateway connection failures or user authentication failures returned by the Authentication Gateway's STS. This level of information will typically be found in the instance's log files. If an access error (404/403) is returned to the PAS for OpenEdge client then look into the instance's log files for errors when checking the URL access controls against the STS issued Client-Principal Role attribute.

Advanced troubleshooting can be performed by enabling DEBUG logging in the web application's `WEB-INF/logging.xml` file. DEBUG settings that are specific to STS authentication are:

```
<logger name="com.progress.appserv.services.security.OESTSAuthProvider" level="DEBUG"/>
<logger name="com.progress.appserv.services.security.OESTSUserDetailsImpl"
level="DEBUG"/>
```

Other Spring Security authentication and URL authorization process errors can be discovered by enabling DEBUG logging:

```
<loggername="org.springframework [optional-package-path ]" level="DEBUG" />
```

OpenEdge Authentication Gateway utilities

The following sections contain information about the command line utilities used for managing and configuring the OpenEdge Authentication Gateway:

- Change the keystore password utility (changeP12pwd) on page 146
- Data export utility on page 147
- STS Client Utility (stsclientutil) on page 148
- STS Connection Role Utility (stsconnroleutil) on page 156
- Strong Password Encryption Utility (stspwdutil) on page 166

Change the keystore password utility (changeP12pwd)

This utility changes the keystore password in a specified keystore file. The utility is found in the `/conf` folder of a PAS for OpenEdge instance.

The default password is `password` in the default keystore file: `%DLC%/bin/tomcat-keystore.p12`. Obviously, the password should be changed when you are deploying a production instance of PAS for OpenEdge.

Note:

You can check the current password by running: `tcman config psc.as.https.keypass`. After changing the password with `changeP12pwd`, set `psc.as.https.keypass` to the new password. (See the example below.)

Syntax

```
changeP12pw { -keystorefile file_name –keypassword
current_password } | -h
```

Parameters

-file_name

Specifies the file name (or complete path) of the keystore file.

-current_password

Specifies the existing password in the keystore file.

-h

Displays a list of the changeP12pwd parameters.

Example

The following example changes the default password. The changeP12pwd utility is being invoked from the /conf directory on the default keystore file that is in the same directory.

```
proenv>changeP12pwd  -keystorefile  tomcat-keystore.p12 -keypassword password
MAC verified OK
##### Take backup of original tomcat-keystore.p12 as tomcat-keystore_bk.p12
#####.alias ############## : test2
Loading 'screen' into random state - done
Enter Export Password: newpassword
Verifying - Enter Export Password:  newpassword
"### Success: tomcat-keystore.p12 password changed ###"
```

Notice that:

- The utility creates a backup of the original keystore file. In this case, it is tomcat-keystore_bk.p12.
- The utility prompts you to enter the new password and to verify the entry.
- The utility indicates if the change was successful.

After a successful password change, reset psc.as.https.keypass to the new value:

```
tcman config psc.as.https.keypass=newpassword
```

Data export utility

This utility is an ABL procedure that provides an administrator with a tool that can output Domain records from an OpenEdge database. This output can be used as input for an STS application's Domain configuration file and for the Domain keystore file generator (gendomreg).

The procedure writes two log files, which are created in the directory specified by the FOLDER parameter:

- domain_dump_runner.log — a log file for the procedure (independent of the data export)
- domain_dump.log — a log file for the actual data export

Syntax

```
_progres -p "OpenEdge/DataAdmin/Util/dump_domains.p"
         -db db-name [ -db db-name [ . . . ]
] -param "PARAM_NAME:VALUE  [,PARAM_NAME:
VALUE  [,. . .]  ]"
```

Parameters

The following table contains a description of the supported parameters and their valid values:

Parameter	Valid values	Notes
LOG	Any of the names in OpenEdge.Core.LogLevelEnum	If no value is passed in, or the value is invalid (no in the enum), the WARN level is used.
FOLDER	An existing folder into which data is exported and log files are written	If no value is specified, or if the value is invalid (i.e. the folder does not exist), the WRKDIR environment variable is queried and if not found, the SESSION:TEMP-DIR is used
MAKESSO	A logical TRUE/FALSE value	The ABL LOGICAL( ) function is used to evaluate the value. The default is FALSE if no value is specified or if the value doesn't resolve to true or false.
CONFIG-FILE	A filename that will be created	Used for the domain configuration file. Sets the export utility's DomainConfigFile property.
KEYSTORE-FILE	A filename that will be created	Used for the domain keystore. Sets the export utility's DomainKeystoreFile property.

STS Client Utility (stsclientutil)

The stsclientutil utility is an administrator-level troubleshooting utility used in troubleshooting OpenEdge database connections to the STS application. Specifically stsclientutil addresses these tasks:

- Establish the OpenEdge installation's ability to connect to a PAS for OE server and use its STS application (i.e. *ping* the STS).
- Test the ability of the OpenEdge installation to use an STS application for direct user login operation for a specific OpenEdge Domain (i.e. *authenticate* the user).
- Test the ability of the OpenEdge installation to use an STS application for operating system user login SSO to a specific OpenEdge Domain (i.e. *exchange* the OS security token for an OpenEdge security token).

```
stsclientutil -cmd command -url sts-url
       [command-options]
```

-cmd *command*

Specify the task to perform from one of the following:

- **ping** — Send an empty message to an STS application to test HTTPS connections and STS Key features.
- **authenticate** — Send user login credentials to the STS application to test its Domain configuration.

  With authenticate, you must also specify the OpenEdge user-id used to send to the STS application for direct user login testing, as shown:

  -user *user* [@*domain*]

- **exchange** — Send the security token of the current process's operating system user login to the STS application to test its Domain configuration

  With exchange, you must also specify the OpenEdge domain name to send to the STS application for operating system SSO testing, as shown:

  -domain *domain*

Note: There is no default for *command*, one of the options must be specified.

-url *sts-url*

Specify the URL of the STS application (secure PAS for OE server that includes a Security-Token-Service (STS) application) used by the OpenEdge database. There is no default for *sts-url*, the URL must be specified.

command-options

The following options are available for each of the -cmd choices:

-sslversion { TLSv1 | SSLv2 | SSLv3 | TLSv1.0 | TLSv1.1 | TLSv1.2 | TLSv1.3 }

Specify TLS protocol to use when connecting to the PAS for OpenEdge server hosting the STS application. If not specified, the default is TLSv1.2.

-sslciphers *cipher* [,*cipher*...]

Specify one or a list of TLS cipher-suites to use when connecting to the PAS for OpenEdge server hosting the STS application. If not specified, the default is TLSv1.2 for all ciphers.

-installpath *dlc-path*

Specify the operating system file path of the OpenEdge installation to test, in the event of multiple OpenEdge installations. If not specified, the default is $DLC or %DLC%.

-keystorepath *ks-path*

Specify the operating system file path to where the STS Client Key is installed. If not specified, the default is $DLC/keys.

The STS Client Key is installed by a DBA who has physical access to the OpenEdge Authentication Gateway Server Key. The key is managed with the stskeyutil utility.

`-certstorepath` *cs-path*

Specify the operating system file path to where the HTTPS (TLS) CA certificates can be found to validate the PAS for OpenEdge server's certificate. If not specified, the default is `$DLC/certs`.

`-logginglevel` *level*

Specify the amount of `stdout` logging to be provided during troubleshooting activities. If not specified, the default level is 2. The range is 0-5.

`-nohostverify`

Specify this option to suppress the validation of the PAS for OpenEdge certificate's subject name against the URL host DNS name.

The host name checking of TLS server certificates compares the name returned by a DNS lookup of the URL's *host* field to the *CN* subfield of the server certificate's *subject-name* X509 name.

`-servername`

Use this parameter when the OpenEdge Authentication Gateway server is configured with multiple virtual hosts that are bound to a single IP address. Specify the hostname that you want the STS client utility to connect to. The STS client utility requests the virtual host's SSL certificate during the SSL handshake instead of the Server URL's host.

The `stsclientutil` utility is a secured application that is capable of being included into DBA automation scripts without fear of disclosing sensitive user credentials or OpenEdge security tokens that could be used to gain access to OpenEdge databases.

Examples

Examples of `stsclientutil` with the `ping` command:

```
stsclientutil -cmd ping -url https://sts.acme.com:8992

stsclientutil -cmd ping -url https://sts.acme.com:8992/oests
```

Examples of `stsclientutil` with the `authentication` command:

```
stsclientutil -cmd authentication -url https://sts.acme.com:8992 -user fred -nohostverify

stsclientutil -cmd authentication -url https://sts.acme.com:8992 -user 'fred@acme.admins'

stsclientutil -cmd authentication -url https://sts.acme.com:8992 -user 'fred@acme.admins'
 -sslversion TLSv1.0
```

Examples of `stsclientutil` with the `exchange` command:

```
stsclientutil -cmd exchange -url https://sts.acme.com:8992 -domain 'acme.osusers'

stsclientutil -cmd exchange -url https://sts.acme.com:8992 -domain 'acme.osusers'
-logginglevel 5
```

STS Key Utility (stskeyutil)

The `stskeyutil` command provides you with the capability to create and manage server and client keys. Major options include:

- `create`
- `verify`
- `install`
- `keyfile`

Testing options include:

- `keytest`
- `headertest`
- `ssotest`

The syntax of the specific commands is discussed in the sections that follow.

stskeyutil common options

All of the `stskeyutil` commands support the following common options:

```
-file filename
-keystorepath path
-silent
-url sts-url
-v
```

`-file` *`filename`*

Specify the STS server key file name. The default is `oests-key.ecp`.

`-keystorepath` *`path`*

Identify the directory location of the client file. The default is `%DLC%/keys`.

`-silent`

Do not prompt for password, access-codes, or file paths.

`-url` *`sts-url`*

Specify the STS server's url in the format: `https://<host>[:port]/[web-app]`.

`-v`

Provide additional run-time information as the utility executes (verbose).

Common options are represented by **[** *`common-options`* **]** in the utility syntax box.

stskeyutil create utility

Generates an STS Server Key file.

```
stskeyutil create
     [ -overwrite
     | -header head
     | -user usr
     | -roles roles
     | -domain name
     | -expires n
     ]
     [ common-options ]
```

`-overwrite`

Overwrite and existing key file. By default, an existing key file is not overwritten.

`-url` *`url`*

This is one of the common options that specifies the OpenEdge Authentication Gateway server's URL into the key. If not specified, no URL is written to the key. This does not affect the ability to use the key.

`-header` *`head`*

Specify a client STS Client Key in *`head`* to send to the to Authentication Gateway Server in the header. If not specified, the client uses the default header name (x-oests-token).

`-user` *`usr`*

Specify the Client-Prinicipal user-id in *`usr`*. If not specified, the default user-id is `oeclient`.

`-roles` *`roles`*

Assign roles for authorization to the OpenEdge STS resources. If not specified, the default role is `OESTSClient`.

`-expires` *`n`*

The amount of time before the key expires. Specify the number of days in *`n`*.

The `stskeyutil create` utility generates a sealed key file containing a sealed OpenEdge Client-Principal, using an administrator-supplied Domain access code. The Client-Principal field contents generated for the STS Server Key is not controllable by the administrator.

Note: When the OpenEdge STS server is running in a clustered environment, the administrator does not need to create an STS Server Key for each node. Assuming all nodes in the cluster share the same OpenEdge STS Server configuration, they share the STS Server Key. Note that when installing the STS Client Key, you can specify a different node name via the `-node` parameter.

---

stskeyutil verify utility

Verifies access to and the integrity of an STS Server Key file.

```
stskeyutil verify
    [ -seal | -info ]
    [ common-options ]
```

`-seal`

Specify to validate the STS server key's Domain Access Code.

`-info`

Specify to display server key information.

The `stskeyutil verify` utility password decrypts the *STS Server Key* file using an administrator supplied password and verifies its contents. (You can optionally verify the domains access code as well via the -seal parameter.) Successful verification yields the following message:

```
Server key validation success
```

Note: Adding `-info` to verbose tracing outputs certain key information fields for verification purposes. However, certain elements of the key file are not visible to the user so that the key cannot be fully generated from the information displayed.

stskeyutil install utility

Creates a new STS Client Key file from an STS Server Key file.

```
stskeyutil install -url sts-url
    [ -overwrite ]
    | -node node-name]
    [ common-options ]
```

`-url` *`sts-url`*

Specify the STS server's url in the format: `https://<host>[:port]/[web-app].`

`-overwrite`

Overwrite an existing key file. By default, an existing key file is not overwritten.

`-node` *`node-name`*

Specify the cluster node named *`node-name`*.

The `stskeyutil install` creates and installs an STS Client Key that corresponds to a specific OpenEdge STS server key. The resulting STS Client Key is encrypted and stored inside the OpenEdge installation's (default) `%DLC%/keys` directory.

Note: Each OpenEdge STS server used by an OpenEdge installation's products requires that a separate STS Client Key be created and installed using the STS Server Key. This requires the administrator to create the key by running `stskeyutil install` on the host where the STS Client Key file will be written and from within the OpenEdge installation the STS Client Key is generated for.

Each installed STS Client Key file is bound to a single OpenEdge installation, on a single server, and to the OpenEdge STS (URL) used for access. The URL must match exactly how it is set up on the database configuration. This binding is reflected by the hashed file name of the STS Client Key file.

When the OpenEdge STS server is running in a clustered environment it may not always be possible to generate each node's STS Client Key by logging directly into each node and executing the install command. To enable generating a STS Client Key for each cluster node, an administrator may follow this process:

- Ensure that each cluster node will use the same exact absolute path to the OpenEdge installation.

  Warning: If this is not true, stop now!

- Obtain each cluster node's node-name (found using the UNIX `uname -n`)
- Log in to the active cluster node as the OpenEdge administrator, and execute `stskeyutil install` for the current cluster node
- Repeat, executing `stskeyutil install` once for each additional cluster node, adding `-node` *`node-name`* to the command line

stskeyutil keyfile utility

Display the full path to a STS client key file.

```
stskeyutil keyfile -url sts-url
       [ -node node-name ]
       [ common-options ]
```

`-url` *`sts-url`*

Specify the STS server's url in the format: `https://<host>[:port]/[web-app]`.

`-node` *`node-name`*

Display the *STS Client Key* filepath for the cluster node named *`node-name`*.

The `stskeyutil keyfile` displays the full path to the STS client key file based on the URL of the OpenEdge STS Server (the default, or one you supply on the command line). Output to `stdout` shows the file path information.

Note:

When the OpenEdge STS server is running in a clustered environment, it may not always be possible to log into each node in order to obtain it's STS Client Key file name by executing the keyfile command. To enable discovering each cluster node's STS Client Key filename, follow this process:

- Obtain each cluster node's node-name (found using the UNIX `uname -n`)
- Log in to the active cluster node as the OpenEdge administrator
- Execute `stskeyutil keyfile` for the current cluster node
- Repeat, executing `stskeyutil keyfile` once for each additional cluster node adding `-node` *`node-name`* to the command line

stskeyutil keytest utility

Tests access to an existing STS Client Key file.

```
stskeyutil keytest -url sts-url
    [ -info | -seal -file name ]
    [ -node node-name ]
    [ common-options ]
```

`-url` *`sts-url`*

Specify the STS server's url in the format: `https://<host>[:port]/[web-app]`.

`-info`

Display client key information.

`-seal -file` *`name`*

Validate the client STS key's domain access code, by supplying the server STS key file in *`name`*.

`-node` *`node-name`*

Display the *STS Client Key* filepath for the cluster node named *`node-name`*.

An administrator may test an OpenEdge installation's ability to decrypt and use an STS Client Key file using `stskeyutil keytest`. Run this utility, entering the OpenEdge STS server's URL on the command line. Output to `stdout` displays success or failure information.

When the OpenEdge STS server is running in a clustered environment it may not always be possible to test each node's STS Client Key by logging directly into each node and executing `stskeyutil keytest`. To enable testing a STS Client Key for each cluster node an administrator may follow this process:

- Obtain each cluster node's node-name (found using the UNIX `uname -n`)
- Log in to the active cluster node as the OpenEdge administrator, and execute`stskeyutil keytest` for the current cluster node
- Execute `stskeyutil keytest` once for each additional cluster node, adding `-node` *`node-name`* to the command line

stskeyutil headertest utility

Test the use of a STS Client Key to produce an HTTP header for use in gaining authorization to an OpenEdge STS server.

```
stskeyutil headertest -url url
    [ -info ]
    [ common-options ]
```

`-url` *url*

Specifies the OpenEdge Authentication Gateway server's URL into the key. If not specified, no URL is written to the key.

`-info`

Display client key information.

Note: To perform`stskeyutil keytest` in a cluster's environment, it must be executed in the node the OpenEdge database and/or ABL client executes in. That is, the `-node` parameter is not supported.

---

stskeyutil ssotest utility

Test the use of an STS Client Key to produce an OS user-id token that may be validated by an OpenEdge STS server and exchanged for a full OpenEdge Client-Principal token.

Note: To perform `stskeyutil headertest` in a cluster's environment, it must be executed in the node the OpenEdge database and/or ABL client executes in. That is, the `-node` parameter is not supported.

```
stskeyutil ssotest -url url
    [ common-options ]
```

`-url` *url*

Specifies the OpenEdge Authentication Gateway server's URL into the key..

STS Connection Role Utility (stsconnroleutil)

As part of using the OpenEdge Authentication Gateway, database connection role authorisation is added. This allows DBA's to restrict the set of users who may connect to a database at any given time.The feature is enabled via a record in the `_db-options` table of the database that is checked when the use of the gateway is enabled. A default role called `_db.connection` is used to grant individual users access to the database. No wildcard grants are used.

The stsconnroleutil utility provides a DBA the ability to control which user accounts are granted membership in a database's connection role from the operating system command line. The connection role is used to grant/revoke an individual user's right to either connect to their database, or be set as the connection's current user. Options include:

- status
- enable
- disable
- list
- grantuser
- grantfile
- revokeuser
- revokefile

The syntax of the specific commands is discussed in the sections that follow.

stsconnroleutil status utility

Display the status connection role for the specified database.

```
stsconnroleutil status
     [ -folder folder-name]
     -db dbname [db-options]
          [-db dbname [db-options] ... ]
     [abl-startup-options]
```

`-folder` *folder-name*

Specify a directory (folder) where log files and temporary files are written. If not provided, the following locations are tested, and the first available folder is used:

- The directory specified by the WRKDIR environment variable
- Current working directory
- The session temp-dir specified using -T

`-db dbname [` *db-options* `]`

Specify one or more OpenEdge databases to connect to. *db-options* allows you to specify additional connection parameters.

`[-db` *dbname* `[` *db-options* `] ... ]`

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

The `stsconnroleutil status` utility writes the status of the database connection roles to the file, `dbconnection_status.txt`. One line per database is written to the file in the format:

```
DbName,Enabled,RoleName
```

stsconnroleutil enable utility

Enables the *connection role* feature in each specified database.

```
stsconnroleutil enable
    [ -admin admin-username]
    [ -folder folder-name]
    -db dbname [db-options]
        [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-admin` *admin-username*

Specify the administrator username. If not specified, the current user is assumed to be the administrator.

`-folder` *folder-name*

Specify a directory (folder) where log files and temporary files are written. If not provided, the following locations are tested, and the first available folder is used:

- The directory specified by the `WRKDIR` environment variable
- Current working directory
- The session temp-dir specified using `-T`

`-db` *dbname* **[** *db-options* **]**

Specify one or more OpenEdge databases to connect to. *db-options* allows you to specify additional connection parameters.

[`-db` *dbname* **[** *db-options*`]` `...` **]**

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

stsconnroleutil disable utility

Disables the *connection role* feature in each specified database.

```
stsconnroleutil disable
    [ -folder folder-name]
    -db dbname [db-options]
        [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-folder` *folder-name*

Specify a directory (folder) where log files and temporary files are written. If not provided, the following locations are tested, and the first available folder is used:

- The directory specified by the `WRKDIR` environment variable
- Current working directory
- The session temp-dir specified using `-T`

`-db` *dbname* **[** *db-options* **]**

Specify one or more OpenEdge databases to connect to. *db-options* allows you to specify additional connection parameters.

[`-db` *dbname* **[***db-options***]** `...` **]**

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

stsconnroleutil list utility

List the current user's granted *connection role* membership for each specified database.

```
stsconnroleutil list
    [ -folder folder-name]
    -db dbname [db-options]
        [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-folder` *folder-name*

Specify a directory (folder) where log files and temporary files are written. If not provided, the following locations are tested, and the first available folder is used:

- The directory specified by the `WRKDIR` environment variable
- Current working directory

- The session temp-dir specified using -T

-db dbname [*db-options*]

Specify one or more OpenEdge databases to connect to. *db-options* allows you to specify additional connection parameters.

[-db *dbname* [*db-options*] ...]

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

The stsconnroleutil list utility writes the database connection roles to the file, dbconnection_list.txt. One line per grant is written to the file in the format:

DbName,Grantee,CanGrant,Grantor,RoleName.

stsconnroleutil grantuser utility

Grants the specified user membership in the connection role of each specified database.

```
stsconnroleutil grantuser
     -user username -can{ yes | no }
    [ -folder folder-name]
    -db dbname [db-options]
         [-db dbname [db-options] ... ]
    [abl-startup-options]
```

-grantee *username*

Specify the user to grant connection role membership

-can{ yes | no }

Specify if the user being granted connection role membership can grant membership to other users. You can also specify true or false.

-folder *folder-name*

Specify a directory (folder) where log files and temporary files are written. If not provided, the following locations are tested, and the first available folder is used:

- The directory specified by the WRKDIR environment variable
- Current working directory
- The session temp-dir specified using -T

`-db` *dbname* [*db-options*]

Specify one or more OpenEdge databases to connect to. *db-options* allows you to specify additional connection parameters.

[`-db` *dbname* [*db-options*] ...]

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

stsconnroleutil grantfile utility

Grants each user in the specified file membership in the connection role of each specified database.

```
stsconnroleutil grantfile -file filename
    [ -folder folder-name]
   -db dbname [db-options]
        [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-file` *filename*

Specify a file containing qualified user-ids to grant connection role membership. The format of the file is as follows:

- One user-id per line, with comma-delimited values
- The first field is the user-id
- The second field is the can-grant value (yes or no), indicating if the user can grant connection role membership to others.

`-folder` *folder-name*

Specify a directory (folder) where log files and temporary files are written. If not provided, the following locations are tested, and the first available folder is used:

- The directory specified by the `WRKDIR` environment variable
- Current working directory
- The session temp-dir specified using `-T`

`-db dbname` [*db-options*]

Specify one or more OpenEdge databases to connect to. *db-options* allows you to specify additional connection parameters.

[`-db` *dbname* [*db-options*] ...]

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

stsconnroleutil revokefile utility

Revokes membership in the connection role for each user in the specified file of each specified database.

```
stsconnroleutil revokefile -file filename
    [ -folder folder-name]
    -db dbname [db-options]
        [-db dbname [db-options] ... ]
    [abl-startup-options]
```

-file *filename*

Specify a file containing qualified user-ids to revoke connection role membership. The format of the file is one user-id per line.

-folder *folder-name*

Specify a directory (folder) where log files and temporary files are written. If not provided, the following locations are tested, and the first available folder is used:

- The directory specified by the WRKDIR environment variable
- Current working directory
- The session temp-dir specified using -T

-db *dbname* [*db-options*]

Specify one or more OpenEdge databases to connect to. *db-options* allows you to specify additional connection parameters.

[-db *dbname* [*db-options*] ...]

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

stsconnroleutil revokeuser utility

Revokes membership in the connection role for the specified user of each specified database.

```
stsconnroleutil revokeuser
     -user username
    [ -folder folder-name]
    -db dbname [db-options]
         [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-grantee username`

Specify the user to revoke connection role membership

`-folder folder-name`

Specify a directory (folder) where log files and temporary files are written. If not provided, the following locations are tested, and the first available folder is used:

- The directory specified by the `WRKDIR` environment variable
- Current working directory
- The session temp-dir specified using `-T`

`-db dbname [ db-options ]`

Specify one or more OpenEdge databases to connect to. *db-options* allows you to specify additional connection parameters.

`[-db dbname [db-options] ... ]`

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

STS URL Utility

The `stsurlutil` utility provides you with the capability to control which OpenEdge Authentication Gateway server a database will use to authenticate users during a connection or switch-user. Options include:

- `status`
- `list`
- `update`

As part of using the OpenEdge Authentication Gateway, database administrators need to provide an HTTPS URL. This value is held in a record in the `_db-options` table that is checked when the database is enabled to operate with the OpenEdge Authentication Gateway. Only valid HTTPS URLs are allowed, and none that point directly to the localhost. The syntax of the specific commands is discussed in the sections that follow.

stsurlutil list utility

List the current OpenEdge Authentication Gateway server URL for each specified database.

```
stsurlutil list
     [ -folder folder-name]
     -db dbname [db-options]
          [-db dbname [db-options] ... ]
     [abl-startup-options]
```

`-folder` *folder-name*

Specify a directory (folder) where log files and temporary files are written. If not provided, the following locations are tested, and the first available folder is used:

- The directory specified by the `WRKDIR` environment variable
- Current working directory
- The session temp-dir specified using `-T`

`-db dbname [` *db-options* `]`

Specify one or more OpenEdge databases to connect to. *db-options* allows you to specify additional connection parameters.

`[-db` *dbname* `[`*db-options*`] ... ]`

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

The `stskeyutil list` utility writes the gateway URL to the file, `dbauthgateway_list.txt`. One line per database is written to the file in the format:

```
DbName,URL
```

stsurlutil status utility

Display the status of the OpenEdge Authentication Gateway for each specified database.

```
stsurlutil status
     [ -folder folder-name]
     -db dbname [db-options]
          [-db dbname [db-options] ... ]
     [abl-startup-options]
```

`-folder` *folder-name*

Specify a directory (folder) where log files and temporary files are written. If not provided, the following locations are tested, and the first available folder is used:

- The directory specified by the `WRKDIR` environment variable
- Current working directory
- The session temp-dir specified using `-T`

`-db dbname [` *db-options* `]`

Specify one or more OpenEdge databases to connect to. *db-options* allows you to specify additional connection parameters.

`[-db` *dbname* `[`*db-options*`] ... ]`

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

The `stsurlutil status` utility writes the status of the gateway URL to the file, `dbauthgateway_status.txt`. One line per database is written to the file in the format:

```
DbName,Enabled
```

stsurlutil update utility

Change the OpenEdge Authentication Gateway server URL for each specified database.

Note: The change takes effect the next time the database is started.

```
stsurlutil update -url url
     [ -servername virtual-hostname]
     [ -folder folder-name]
     [-ssl args]
     [-test true | false]
     -db dbname [db-options]
          [-db dbname [db-options] ... ]
     [abl-startup-options]
```

`-url` *url*

Specify the new URL for the Authentication Gateway to insert into the database configuration.

Specify *url* using the format: `https://<host>[port][/sts-application-name]`

`-servername` *virtual-hostname*

Use this optional parameter when the Authentication Gateway is configured with multiple virtual hosts that are bound to a single IP address. Specify the hostname that you want the database to connect to. This creates a row in the database's options table (`_db-option`) that stores the virtual hostname. At runtime, when the database needs to connect to the Authentication Gateway, it requests the virtual host's SSL certificate during the SSL handshake instead of the server URL's host.

`-folder` *folder-name*

Specify a directory (folder) where log files and temporary files are written. If not provided, the following locations are tested, and the first available folder is used:

- The directory specified by the `WRKDIR` environment variable
- Current working directory
- The session temp-dir specified using `-T`

`-ssl` *args*

Additional TLS (https) connection options. (See the ABL socket:connect method.)

`-test true` | `false`

Tests if the URL is reachable (connect-only.) The default is `true`, if not specified. Only the literal values `true` and `false` are accepted (no other traditional boolean pairs).

`-db dbname` [*db-options*]

Specify one or more OpenEdge databases to connect to. *db-options* allows you to specify additional connection parameters.

[`-db` *dbname* [*db-options*] ...]

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

The `stskeyutil update` utility writes the new URL of the gateway to the database.

Strong Password Encryption Utility (stspwdutil)

The `stspwdutil` utility generates strong (AES-128) encrypted passwords for inclusion into configuration storage and verifies an encrypted password against a clear text string.

```
stspwdutil
      { encrypt text-input [ key-pad-id ]
          |
            verify encrypted-string reference-string }
```

text-input

Specify a clear-text value to encrypt and return as a hex-binary encoded value.

Use operating system shell command line escapes and quoting to enter meta-character text.

key-pad-id

Specify an alternate Key Pad ID. The default Key Pad ID is zero (0), and valid values are the range of 0 - 9.

encrypted-string

Specify the hex-binary encoded value to verify.

reference-string

Specify a clear-text value to use in the validation process of *encrypted-string*. If you do not supply a *reference-string* value, `stspwdutil` returns an error.

When `stspwdutil` is executed with the `encrypt` option, it takes as input a clear text string and an optional Key Pad ID. The output is an encrypted string.

When `stspwdutil` is executed with the `verify` option, it takes as input an encrypted string, and a clear text string to use as a base to verify the encrypted string. The output is `true` or `false`.

The `stspwdutil` utility generates strong (AES-128) encrypted passwords. The output uses an encryption prefix whose algorithm type `'oep'` and an encoding type of `'h'` (for hex-binary encoding), and a Key Pad ID of 0-9. The encoded hex-binary is in blocks of 16 characters, which reflects its AES encryption algorithm's block size. So the encoded value's size does not reflect the actual size of the encrypted text data.

The encryption prefix of encrypted text has the form `aaaed::`, described as follows:

`aaa` — Designates the encryption algorithm type code

`e` — Designates the text encoding used for the encrypted value, `h` is specified for hex-binary

`d` — Designates the Key Pad ID used to generate an encryption key

`::` — The delimiter between the prefix and the encoded data

Examples

Examples of `stspwdutil` with the `encrypt` option:

```
proenv> stspwdutil encrypt password
oeph0::76E5F6C162276768465F02E4D2D1DDCD


proenv> stspwdutil encrypt password 3
oeph3::3B295B1855077677CB2ED521DD27C60D
```

Examples of `stsclientutil` with the `verify` option:

```
proenv> stspwdutil verify eph0::76E5F6C162276768465F02E4D2D1DDCD password
true


proenv> stspwdutil verify eph0::76E5F6C162276768465F02E4D2D1DDCD mypasswd
false
```

Authentication using OAuth2 and JWT

OAuth (Open Authorization) is a standard framework that allows login access to third-party websites and applications without exposing user account credentials and information. Currently, OAuth2 is the latest version of that standard.

When a product is comprised of components from multiple sources that must share a common user identity, OAuth2 and JSON WebTokens(JWT) provide the means to support single user authentication to the product as a whole. Authentication results in a Single Sign-On (SSO) security token that allows access to all the product services that are enabled to use the token for authorization.

OAuth2, implemented correctly, provides a medium level of security for a broad range of web application architectures, including web browsers, mobile devices, browser-based client applications (such as JavaScript), and B2B client applications.

OAuth2 and JWT standards are enabled in PAS for OpenEdge to allow it to participate in SSO outside a homogeneous OpenEdge environment. This section provides a general description of the OAuth2 and JWT standards, a description of how the standards are supported in PAS for OpenEdge's Spring Security framework, and, instructions on configuring and troubleshooting.

Topics include:

- OAuth2 Concepts and Terms on page 168
- OAuth2 Security Considerations on page 170
- OAuth2 Tokens on page 170
- JSON Web Tokens (JWT) on page 172
- Support for OAuth2 and JWT in PAS for OpenEdge on page 175
- Configuring a PASOE Web Application as an OAuth2 Resource Server on page 177
- Debug Logging for OAuth2 on page 184

OAuth2 Concepts and Terms

OAuth (Open Authorization) is a standard framework that allows login access to third-party websites and applications without exposing user account credentials and information. Currently, OAuth2 is the latest version of that standard.

OAuth2, implemented correctly, provides a medium level of security for a broad range of web application architectures, including web browsers, mobile devices, browser-based client applications (such as JavaScript), and B2B client applications.

This section will introduce core OAuth2 concepts and terms.

Actors

OAuth2 defines the following *actors* to perform specific operations and to interact to create a secure authorization framework:

Actor	Description	OpenEdge Examples
Resource Owner	A user who is the owner of data stored, accessed through, and protected by, a *Resource Server*. As data owners, they can authorize a *Client* to access and perform certain operations on data stored on a *Resource Server*.	End-users
Client	An application that uses an access token from an *Authorization Server* to gain access to the *Resource Owner* data that is hosted by a *Resource Server.* Each *Client* is defined by a unique ID (`client_id`) that is registered with an *Authorization Server*, inserted into an OAuth2 *Access Token*, and validated by the *Resource Server*.	JavaScript client applications
Resource Server	The web server data service that hosts and authorizes *Client* access to a *Resource Owner*'s data based on an *Access Token*'s client_id, user-identity and granted scope.	ABL business applications (Spring OAuth2 in OEABL web applications)
Authorization Server	A server that validates a *Resource Owner*'s credentials. Maintaining a list of registered *Clients* and *Resource Servers*, it calls on an *Authentication Server* for identity verification. If authorization is successful, it issues access tokens to a *Client* on behalf of the *Resource Owner*.	N/A (3rd party products)
User-agent	A web browser, or some other type of framework that hosts *Client* applications.	Browsers, mobile devices, Node js, etc.
Authentication Server	A server, undefined in the standard, that is used by an *Authorization Server* to authenticate the *Resource Owner*'s identity. Oncer the *Resource Owner*'s identity is authenticated, the *Authorization Server* can continue its authorization process for issuing an access token.	N/A (3rd party products associated with the *Authorization Server*)

Flows and Grant Types

A*Flow* represents an exchange of one or more HTTP messages between the various OAuth2 Actors. Each OAuth2 *Flow* has a name defined in the standard as a *Grant Type*. The Grant Type is configured and implemented in a Client and a Authorization Server, and may be included in an access token for validation by a Resource Server.

The Grant Types currently specified include:

- Authorization (Code) Grant

  A flow used by confidential clients. The Client interacts with the Resource Owner and interfaces with an Authentication Server via a User-agent. The Resource Owner authenticates to an Authorization Server who issues an Authorization code to the Client. The Client then uses the Authorization Server to exchange its Authorization code for an access token it can use to access the Resource Server on behalf of the Resource Owner

- Resource Owner Password (Credentials) Grant

A flow used in cases where the Resource Owner has a trust relationship with the Client. An example of this type of relationship is a user (Resource Owner) with a mobile device (Client). In this flow, the Client and Resource Owner work together to authenticate to an Authorization Server who issues an access token to the Client. The Client manages the access token and uses it to access the Resource Owner's data on the Resource Server.

- Client Credentials Grant

  A flow that is used only when the Client is a trusted application. For example the Client may be a partner company's business application who is also a Resource Owner. This is a very simple flow where the Client authenticates to the Authorization Server using its own user-id and password and receives an access token in response. The Client uses the access token to access data on a Resource Server

  .

- Implicit Grant

  A flow often used by a public Client. In this flow, the Client initiates the flow by directing a User-agent to an Authorization Server who interacts with and authenticates the Resource Owner. If the Resource Owner grants the Client access to their data, the User-agent is sent an access token it can use on behalf of the Client when the Client accesses the Resource Server.

OAuth2 Security Considerations

Some security experts think OAuth2 is not a secure mechanism for use in browsers, JavaScript applications, web servers, or web applications. However, you can use OAuth2 to reduce security vulnerabilities if you follow best practices. The basic best practices are::

- Securing Authorization and Resource Server implementations using code reviewed/scanned implementations
- Ensuring that every HTTP message travels via TLS network connections
- Ensuring that cryptography keys are securely stored and shared between Authorization and Resource Servers
- Fully validating an access token (according to its specifications) before it is used used to access data

OAuth2 Tokens

In OAuth2, security tokens transport authorization information between Clients, User-agents, Authentication Servers, and Resource Servers. Tokens are opaque values in the OAuth2 standard and can be implementer defined. The two actors that must agree on a Token's form and content are the Authorization Server that produces a the token, and the Resources Server that uses the token's value to grant/deny access to Resource Owner data. to a Client.

Note: The server that produces the OAuth2 token is a 3rd party implementation. The tokens cannot be generated from the OpenEdge environment

---

A token must be recognized by a server as one of these forms before it can be used:

- Authorization Token
- Self-contained Access Token (AKA ID token; Identity Token)
- Random Access Token
- Refresh Token

- Bearer Token

Authorization Token

An Authorization Token is a random data value issued by an Authentication Server to a Client who then sends it to an Authorization Server and exchanges it for an Access Token.(self-contained or random).

Self-contained Access Token

An OAuth2 ***Self-contained Access Token*** is a base64 encoded data value that holds the *Resource Owner's* identity and granted access rights (aka scope). This form of Access Token can be used directly by a *Resource Server's* authorization process to grant/reject access to services or data, because it contains the Client identity, Resource Owner identity, and Resource Owner scope necessary to perform authorization. In most implementations the token is a JSON Web Token (see JSON Web Tokens (JWT) on page 172 for more information), but the OAuth2 standard does not specify the token's actual format.

Self-contained Access Tokens are commonly used by Federated Authentication services who exist to authenticate users and issue a single Access Token that may be used by multiple vendor's *Resource Server* services. It is the responsibility of the *Resource Server* to limit which Federated Authentication service(s) it will accept Access Tokens from by validating it before it may be used.

The general guideline states that an *Authorization Server* may issue a Self-contained Access Tokens directly to trusted *Clients* who sends it to the *Resource Server* in each service request as a Bearer Token. The *Resource Server's* authorization process will validate the Self-contained Access Token's authenticity, and then use the token's identity or access right values to grant or reject access and add to the service's audit trail.

Some *Authorization Servers* will support a request from a *Resource Server* to exchange an untrusted Client supplied Random Access Token for a Self-contained Access Token that contains the Resource Owner's identity and granted scope . In such a case the *Resource Server* would log into into the issuing *Authorization Server* and exchange the Random Access Token for a Self-contained Access Token. This is a much more complex, non-standards based, process and is not widely available.

Which method is used by a *Resource Server* to obtain a Self-contained Access Token is determined by the *Authorization Server's* configuration and implementation according to the type of registered client. In both use cases the *Resource Server* must obtain and fully validate a Self-contained Access Token for expiration and data integrity before it will grants access to *Resource Owner* data.

Random Access Token

An OAuth2 ***Random Access Token*** is a unique Base64 encoded random data value that is typically issued issued to untrusted *Clients.* Random Access Tokens are commonly used when an *Authorization Server*coexists with one or more *Resource Server* services in the same product. The randomness of the Access Token's value reduces the risk of the *Resource Owner's* identity and access rights being stolen by the *Client*application code. Random Access Tokens work because the *Authorization Server* AND the *Resource Server* share the same (internet) network and storage of Self-contained Access Tokens.

However, because the Random Access Token does not contain a Client identity, Resource Owner's identity, or granted data rights, a *Resource Server* cannot simply use it to grant access to a Resource Owner's data. Before an exchange of a *Random Access Tokens* can take place, the *Resource Server* and *Authorization Server* must first establish a trust relationship whereby each trusts the other to not compromise any Access Token. The trust relationship can be established a number of different ways, and is not specified as part of the OAuth2 specification. Examples of a trusted relationship between an Authorization and Resource server may be the use of a SQL database, in-memory storage, or a fully authenticated HTTP login.

Refresh Token

An OAuth2 ***Refresh Token*** is a unique Base64 encoded random data value that may [optionally] be issued by an *Authorization Server* in conjunction with a Random/Self-contained Access Token. To minimize the time a Random/Self-contained Access Token may be used after being stolen by a 3rd party, an Access Token's expiration times may be kept very short. To not overly burden the *Resource Owner* with continually re-authenticating in order to obtain a new non-expired Access Token, a *Authorization Server* may be configured to issue a Refresh Token, which the Client application may use to automate the process of obtaining a new Access Token from the *Authorization Server* when the current one has expired.

The automated OAuth2 Refresh Token process would typically take place in the *Client* when the *Resource Server* returns an expired token status. The *Resource Server* only detects Access Token expiration when an expiration value is present in the token and will only return a token expired error to the *Client.* The *Resource Server* does not play a role in the *Client* obtaining a new Access Token. The *Client*, upon detecting an expired token response from the Resource Server will enter into a Refresh flow with the *Authorization Server* to obtain a new Access Token.

Bearer Token

A *Bearer Token* refers to any form of identity token that is sent by a *Client* to a *Resource Server* via an HTTP Authorization header that identifies the authorization-scheme as "Bearer". The token value of the Bearer Token Authorization header is not defined. However, in most cases the value is an OAuth2 Self-contained Access Token, or a simple JSON Web Token (JWT).

Example: "Authorization: Bearer abc6324.dgwyuio269nnnwllllsyyyyy.sssy2ajklyuiowl6789"

JSON Web Tokens (JWT)

A **JWT (JSON Web Token)**, while not part of the OAuth2 standard, is commonly used as the physical structure for a Self-contained Access Token (described above). A JWT holds a Resource Owner's identity, Client identity, issuing & expiration timestamps, and scopes used by Resource Server's authorization process. Other *Authorization Server* defined claims (aka JSON fields) may be added by a vendor's implementation. A JWT is a Base64 encoded value that contains three sub-structures that describe the data-integrity signature algorithm, the user's identity claims, and the data-integrity signature. Each sub-structure is delimited by a period ('.') character.

Structure Name	**Description**
Header	Contains information about the JWT payload's and data integrity signature structures
Payload	Contains multiple JWT claims (JSON fields) that describe an authenticated user's id, issuing & expiration timestamps, scope, client ID, and other values
Signature	Contains a binary digital signature of the Header & Payload structures, that is produced using the signature algorithm type and size contained in the Header

A JWT's payload has a small number of common user identity claim fields, but for the most part is wide-open to customization of what it contains. Common identity claim fields include:

JWT Field Name	**Description**	Required in JWT token	Required in OAuth2 token

iss	The URI of the AuthorizationServer who issued the JWT token	Yes	Yes
sub	The user-id. For OAuth2 it will be the *Resource Owner's* ID	Yes	Yes
aud	The token's audience of consumers. For OAuth2 this will be the *Resource Server's* ID	Yes	Yes
client_id	This identified the unique *Client* ID registered by a *Client* application with an *Authorization Server,* and is used by an *Resource Server* to determine if the issued JWT can be used or not to access *Resource Owner* data	No	Yes
jti	A unique identifier for this token that may be used to detect replay attacks and establish *Client* login sessions	optional	optional
iat	The JWT creation date	Yes	Yes
exp	The JWT expiration date, after which it may not be used by a *Resource Server*	optional	optional
nbf	The JWT validation date, before which it may not be used by a *Resource Server*	optional	optional
scope	OAuth2 field name that carries a space delimited list of '*scopes*' that serve to tell a *Resource Server* what data and operations the *Client* is authorized to access	Yes	Yes

token_type	OAuth2 field name. Indicates the HTTP Authorization header scheme this token was issued in (typically the '*bearer*' value)	optional	optional
<other>	Any other Authorization Server provided claims	optional	optional

A JWT's Header may specify one of a set of data-integrity algorithm types based on the JWS standard, which includes HMAC (secret-key) and RSA (public-private key) types.

Header Field Name	**Description**	**Required**
alg	The JWS algorithm name used for generating and verifying the JWT's Signature field value (See the table of JWS signature names)	Yes
typ	The type of Token data format. If not specified it is often defaulted to JWT by most vendor implementations	No
kid	The alias Key-ID name that identifies a keystore entry that contains the encryption key value used by the *Resource Server* to validate the JWT's Signature field value	No

WS Signature names:

JWS Algorithm Name	**Cryptographic Algorithms**	**Key Size**
HS256	HMAC w. SHA	256
HS384	HMAC w. SHA	384
HS512	HMAC w. SHA	512
RS256	RSA signature w. SHA	256
RS384	RSA signature w. SHA	384
RS512	RSA signature w. SHA	512

Support for OAuth2 and JWT in PAS for OpenEdge

PAS for OpenEdge's support for OAuth2 allows your ABL business application/data service to act as a *Resource Server* that accepts either a OAuth2 Self-contained Access Tokens (in JWT format) or a simple JWT token. The support for JWT and OAuth2 Self-contained Access Tokens relies on having access to the user's identity information in order to generate a Client-Princpal that is usable in your ABL business application to set an OpenEdge database connection's user (and therefore tenancy and audit trail). Only those two types of tokens contain sufficient identity information to create a Client-Principal.

Note: The OAuth2 Random Access Token type is not supported because OpenEdge does not embed or interface with an external Authorization Server that is capable of sharing an OAuth2 Self-contained Access Token with PAS for OpenEdge

As an OAuth2 Resource Server PAS for OpenEdge will execute this process on each HTTP request:

1. Obtain the Authorization header's Bearer token
2. Validate the Bearer token type as either a JWT or OAuth2 Self-contained Access Token (aka a form of JWT)
3. Validate the JWT's signature using the JWT header's algorithm type and the PAS for OpenEdge configured encryption key
4. Validate the JWT's required payload claims
5. If the Bearer token type is a JWT or OAuth2 Self-contained Access Token, validate its required payload claims
6. Authorize the token's user to the PAS for OpenEdge data services using the granted JWT scope claim
7. Create an equivalent OpenEdge Client-Principal that will be delivered to the ABL business application with each request

PAS for OpenEdge’s OAuth2 support is supplied by Spring Security. PAS for OpenEdge extends the core Spring Security OAuth2 project implementation to blend it into the same customer ABL application environment as it does for all other Spring Security authentication and URL data authorization services. That OpenEdge integration includes the formatting of error responses and the creation of Client-Principals that are passed to the ABL business application.

The OAuth2 and JWT standards offer implementation vendors many design and run-time use-cases that result in many configuration properties. The oeableSecurity.properties file found in the oeabl web application's WEB-INF/ directory contains a common subset of those properties and provide you the ability to configure each oeabl web application independently. The full set of configuration properties is found in the PAS for OpenEdge instance's conf/oeablSecurity.properties file and are the default values if they are not found in the oeabl web application's configuration. OpenEdge provides may default property values that will not require changing, however not all properties can contain a useful value and must be configured for each installation.

PAS for OpenEdge will support most OAuth2 or JWT compliant authentication/authorization services. However, there are limitations. An OAuth2 Access Token/JWT must meet this criteria to be usable by PAS for OpenEdge's OAuth2 support:

1. The Access Token/JWT must be received in an Authorization HTTP header with the authorization-scheme equal to "Bearer"
2. The token must contain a user identity (sub) claim to populate the Client-Principal User-ID field
3. The token must contain a scope (scope) claim that can be used by Spring Security to authorize access to the application URLs
4. The token must contain a resource-id (aud) claim that indicates that it may be used by the only those oeabl web application whose resource identity is configured to be of that type

5. An OAuth2 Access Token must contain a client id claim (client_id) that indicates it was issued to a Client type the oeabl web application is configured to support
6. The Access Token/JWT must be received in an Authorization HTTP header with the authorization-scheme equal to "Bearer"
7. The token's signature algorithm must be one of the JWS HMAC or RSA types
8. The encryption key to validate the token's signature field must be available from the issuer of the token and configured
9. If the token issuer adds an expiration claim (exp) then the token must pass the date-time expiration test

The following sections will provide details for configuring a PAS for OpenEdge's OAuth2 support, and is divided into distinct parts:

1. Where to find the configuration files and properties
2. Enabling PAS for OpenEdge's JWT/OAuth2 support
3. Configuring a PAS for OpenEdge Resource Server
4. Configuring JWT Token validation
5. Configuring OAuth2 Token validation
6. Configuring JWT to Client-Principal conversions

Note: Do not proceed without first understanding basic OAuth2, JWT, and OpenEdge Client-Principal terms and functionality.

OAuth2 configuration files

The OAuth2 configuration in PAS for OpenEdge involves these files:

- `oeablSecurity.properties` — contains property settings for JWT / OAuth2 validation operations and Client-Principal generation.

  Note:

  There is a hierarchy of `oeablSecurity.properties` files, which exist in the following locations:

  - *oe_install_dir*`/servers/pasoe/conf/`
  - *instance*`/conf/`
  - *instance*`/ablapps/<abl-app-name>/conf/`
  - *instance*`/<web-app-name>/WEB-INF/`

  See *Progress Application Server for OpenEdge: Administration Guide* for more information.

- `oeablSecurityJWT.csv` — contains the Resource Server’s URL access controls for JWT & OAuth2 Client access based on the token's claim (claim) field
- `oauth2ResSvcClients.cfg` — a file containing OAuth2 Client authorization information for client ID (client_id) field (not required for standard JWT token validation)

Enabling OAuth2 support

OAuth2 support has some fundamental differences that set it apart from other direct user login and OpenEdge SSO processes available in PAS for OpenEdge. Therefore, it has its own Spring Security LoginModel configuration which cannot be combined with any type of direct user-login. .

Note: If multiple forms of client authentication are required for a single OpenEdge ABL application, a second OpenEdge web application should be deployed and mapped to the same ABL application, where the alternate user login may be configured.

Enabling the OpenEdge web application in PAS for OpenEdge to support OAuth2 support is accomplished by setting the `client.login.model` property in the `oeablSecurity.properties` file to `oauth2`.

Note: When `client.login.model` is set to `oauth2`, the `http.all.authmanager` property value has no effect.

Configuring a PASOE Web Application as an OAuth2 Resource Server

Configuring a PAS for OpenEdge web application as an OAuth2 Resource Server requires the following:

- Configuring OAuth2 or JWT Token Validation Services on page 177
- Configuring JWT Signature Field Validation on page 178
- Configuring JWT Payload Field Validation on page 180
- Configuring Self-Contained Access Token Validation on page 181
- Authorizing Access to URLs and HTTP Methods on page 181
- Configuring JWT/OAuth2 Self-contained Access Token conversion to an OpenEdge Client-Principals on page 183

Configuring OAuth2 or JWT Token Validation Services

Setting the Token Services Validation Type

At the core of Spring Security's OAuth2 support is the validation of an access token. The payload of the access token must minimally contain a set of *claim fields*. That minimal list is different depending on whether the token is an OAuth2 Access Token or a simple JWT.

To configure PAS for OpenEdge for the type of token payload validation obtained from the HTTP request's Authorization header, set the following property in `oeablSecurity.properties`:

```
oauth2.resSvc.tokenServices={ jwt | oauth2 }
```

`jwt`

`jwt` tokenServices looks for a JWT in the HTTP request's HTTP Authorization header's *Bearer* scheme value. If a JWT-type Bearer token is found its signature and required JWT payload claims (aud & exp) will be validated before it is passed along to the URL authorization process. Using JWT

tokens has more risk as it exposes the token's information to untrusted Clients, such as a browser in an internet client. This selection is most useful for cases where intranet Clients obtain a JWT from a non-compliant OAuth2 Authorization Server and use it to access a Resource Server without the need for full formal OAuth2 validity checking

`oauth2`

`oauth2` tokenServices incorporates `jwt`T tokenServices and adds additional checking for required OAuth2 standard payload claims. This selection is most useful for cases where either internet or intranet Clients follow one of the four OAuth2 authorization flows to an Authorization Server.

Configuring OAuth2 Sessions

An OAuth2 Resource Server hosts a stateless REST API for its Clients. As a stateless REST API, it will not create user HTTP sessions, but does not preclude the REST API's implementation from creating them. Spring Security provides a property to control the generation and use of HTTP sessions. The default is to follow the normal REST API stateless model.

This can be changed to have Spring Security use and maintain HTTP Sessions by setting the following property's value in `oeablSecurity.properties` to false:

```
oauth2.resSvc.stateless={ true | false }
```

Configuring the WWW-Authenticate Realm Name

An OAuth2 Resource Server accepts Access Tokens via the HTTP Authorization header. If the Resource Server's validation of the Access Token fails it will return a 401 status and a WWW-Authenticate HTTP header with a realm-challenge phrase. The response's realm-challenge phrase can be customized by setting this property value in `oeablSecurity.properties`:

```
oauth2.resSvc.realmName=oeoauth
```

Configuring JWT Signature Field Validation

Validating a JWT Access Token is an essential step in the OAuth2 Resource Server authentication and authorization processes. All JWT access tokens must pass the payload requirements, data integrity signature, and (optionally) expiration. If either of these checks fails the client's HTTP request is rejected.

A JWT's data integrity signature supports multiple types of algorithms. JWT support in PAS for OpenEdge includes HMAC and RSA (public and private) algorithms, as specified in the JSON Web Signature (JWS) standard

Configure the HMAC or RSA type

Choose the JWT signature validation to perform by setting the following property in `openedge.properties`:

```
jwtToken.signatureAlg={ HS256 | HS384 | HS512 | RS256 | RS384 | RS512 }
```

Note: All `HS`*`xxx`* types apply to the HMAC signature algorithm. All `RS`*`xxx`* types apply to the RSA (public and private) signature algorithm.

Configure the HSxxx Signature validation encryption key

The `HS`*`xxx`* encryption key is a single value configured in the `oeablSecurity.properties` file:

```
jwtToken.macKey=value
```

value

A string value, either clear text or encoded text. Encoded text is the output of *oe-install-dir*`/bin/stspwdutil`

Note: Since the encryption key is a single value, PAS for OpenEdge supports only one Authorization Server for each OEABL web application configuration.

Configure the RSxxx Signature validation encryption key

Support for accessing the RSA public keys includes:

- A Java (`.jks`) encrypted keystore file containing a digital certificate holding the single RSA public key
- A file-system directory holding the single PEM-encoded RSA public key
- A JSON Web Key (JWK) set that holds multiple RSA public keys, and I selected by the JWT header's `kid` claim

Configuring a .jks Java keystore

Set the following properties in the `oeablSecurity.properties` file:

```
jwtToken.keystore.type=mac
jwtToken.jksKeyStore=path-to-jks-keystore
```

Note: The Java JKS keystore is maintained by using the Java `keytool` utility. (See https://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html.for more information.)

```
jwtToken.jksKeystore.pwd=value
```

value

A string value, either clear text or encoded text. Encoded text is the output of *oe-install-dir*`/bin/stspwdutil`

```
jwtToken.jksKeystore.alias=sample
```

sample

The name of the keystore entry's alias name used when you store the JWT rsa signature's public-key certificate you obtained from the Authorization Server's operators.

Configuring a PEM encoded RSA public key

All files holding PEM encoded RSA public keys must be in the form *file-name*.pem. You have to configure the file-system directory path (which acts as a keystore) and the alias name to find the PEM file within the file-system directory:

```
jwtToken.keystore.type=pem
jwtToken.pemKeyStore=<file-system-path to PEM files>
jwtToken.pemKeystore.alias=<PEM file-name>
```

Note:

- The PEM-encoded file may optionally contain the RSA private key.
- A sample Pem-encoded file can be found in a PAS for OpenEdge instance's conf/jwtkeys directory.

RSA public/private keys can be generated for testing using the following two command lines:

```
oe-install-dir/bin/sslc genpkey x-out filename.prv.pem -outform PEM -algorithm RSApkeyopt
 rsa_keygein_bits:key-bit-size

oe-install-dir/bin/sslc rsa -pubout -in filename.prv.pem -out filename.pub.pem
```

Configure JWK set holding RSA public keys

A JWK set is a JSON structure that holds one or more individual JWK JSON objects. Each JWK object will contain a single RSA public key and be identified by an alias name. When a JWT is received and its header contains a kid claim, the kid value will be used as the alias name used to lookup a JWK object. If a JWT's header does not contain a kid claim, or a JWK cannot be resolved by alias name, the request is rejected.

```
jwtToken.keystore.type=jwk
jwtToken.jwkKeyStoreURL=URL of where JWK RSA public keys will be loaded from
```

Configuring JWT Payload Field Validation

This section outlines the properties used to validate a JWT Self-contained Access Token's payload fields.

Validating the Resource Owner's ID claim

A JWT has a recommended field name (sub) to hold the authenticated user's ID, but may be configured differently by the issuing Authorization Server. Refer to the JWT issuer's documentation for which field name contains the authenticated user's ID:

```
jwtToken.usernameField=sub
```

Configuring the Resource Server's ID

Each OAuth2 Resource Server has a well-known ID that it registers with the Authorization Server and is used to filter out the JWT tokens issued for other Resource Servers. The Authorization Server will insert a payload audience (`aud`) claim, which will be verified by the PAS for OpenEdge JWT validation process.

To configure the OAuth2 Resource Server ID, change the default `oeablapp` value for the one registered with the Authorization Server:

```
oauth2.resSvc.audience=oeableapp
```

Configuring Self-Contained Access Token Validation

This section outlines the properties used to validate the additional claims found in a Self-contained Access Token's payload.

An OAuth2 Access Token's payload must contain a client_id claim that uniquely identifies each type of Resource Server Client.. You are required to configure a Client ID for PAS for OpenEdge when you have configured the OAuth2 service type to be oauth2:

```
oauth2.resSvc.tokenServices=oauth2
oauth2.resSvc.clientCfg=../oauth2ResSvcClients.cfg
```

The Resource Server's client configuration is normally found in the oeabl web application `WEB-INF/oauth2ResSvcClients.cfg` file, and has the following xml element format:

```
<oauth:client-details-service id="oauth2ResourceClientDetails">
           <oauth:client client-id="oeablClient" />
                ...
      </oauth:client-details-service>
```

Edit the file and change the client-id attribute's value to hold the OAuth2 Client's registered ID.

Authorizing Access to URLs and HTTP Methods

The OAuth2 Access Tokens do not contain user-ids and roles, in the traditional sense, for authorizing access to resources. While a user-id exists in a Self-contained Access Token for logging and audit trail use, it is not used outside of that scope. Instead OAuth2 uses the notion of *client-id* and *scope* to authorize a particular client (application) to use the Resource Server and defines what operations a Resource Owner has authorized the Resource Server to perform for that client.

So the traditional Spring Security Role-Based-Authorization (RBA) and its configuration do not apply to OAuth2. Instead a new form of authorization access control and configuration is used by Spring Security to grant the right to perform an operation.

Basically, OAuth2 authorization is configured using a file named `WEB-INF/oeablSecurityJWT.csv`. It performs the same basic function as the traditional RBA configuration file `oeablSecurity.csv`.

The format of `oeablSecurityJWT.csv` has the same three double-quote fields that hold the URI, HTTP-Method, and access-control specification. The first two fields (URI & HTTP-Method) are the same as in the RBA's `oeablSecurity.csv`. The third field holding the access-control specification uses a different Spring Security expression language.

The expression language used for traditional RBA and OAuth2 scope authorization share the concept of testing whether a security token has been granted a *name*, and that *name* corresponds to granting access to some process, function, or data. A *name* is just a *name*, so long as whoever grants the *name* and the resource server who associates it with access to some resource agree on what the *name* is.

In the case of OAuth2, the *name* is an OAuth2 *scope* that the Authorization Server inserts into the Self-contained Access Token as directed by the Resource Owner. So the same rules apply as with RBA, the Resource Server's configuration has to use the *scope* names configured and used by the Authorization Server. In live production environments, that information will have to be obtained by administrator.

The format of an OAuth2 access-control expression is:

```
access-control-expr := " oauth2-spec [ { or | and } oauth2-spec ... ] "
```

```
oauth2-spec = #oauth2.test( 'value' [, 'value' ... ] )
```

The following table is a list of tests:

clientHasAnyRole('*name*'...)	Check if OAuth2 client has one of the listed Role names
clientHasRole('*name*')	Check if the OAuth2 client as the specified Role name
denyOAuthClient( )	Deny access to all OAuth2 clients, but allow other types of clients
hasAnyScope('*name*' ...)	Check if the OAuth2 client has any named scope specified
hasAnyScopeMatching('*reg-ex*'...)	Check if the OAuth2 client has one of the scopes matching a specified regex expression
hasScope('*name*')	Check if the OAuth2 client has the named scope specified
hasScopeMatching('*reg-ex*')	Check if the OAuth2 client has a named scope matching a specified regex expression
isClient( )	Check if the request is from an OAuth2 client (and not another type of user)
isOAuth( )	Permit only OAuth2 type requests
isUser( )	Check if the current request is from a user and not from an OAuth2 client

Each test returns true/false, and the entire expression must return true in order for the Resource Server to grant access.

Note:

In the default distribution of `oeablSecurityJWT.csv` the default scope names are:

- PSCUser
- PSCAdmin
- PSCDebug

Configuring JWT/OAuth2 Self-contained Access Token conversion to an OpenEdge Client-Principals

Similar to all Spring Security processes in PAS for OpenEdge, a successful authentication and authorization produces an OpenEdge Client-Principal that is delivered to the ABL application code on each request.

The following table indicates the mapping of Self-contained JWT fields to OpenEdge Client-Principal attributes:

JWT Field	Client-Principal Attribute
sub (by configuration)	User-id
exp	expires
scope	Roles
<misc> (if configured)	Properties

Configure mapping the JWT's scope claim to Spring and Client-Principal Roles

The JWT claim field is a comma separated list of values, where each value is mapped to a role name in the Spring and Client-Princpal tokens. Those mapped Role values are used to authorize the client's access to specific URLs:

```
jwtToken.mapScopeToRole={ true | false }
```

A JWT has a recommended field name `scope` to hold the scope of the authenticated user. However, you can configure this field name, as shown below, using the Authorization Server that issues it:

```
jwtToken.scopeNameField={ scope }
```

If the configured field name for scope is not available in the JWT, then the JWT uses `PSCUser` as the default scope. You can set this default scope using the `jwtToken.defaultRoles` property in the `oeablSecurity.properties` file.

Refer to JWT issuer's documentation to find which field name contains the scope of the authenticated user and map its claim to the scope (Role).

Configure JWT claims as Client-Principal property value

JWT tokens may contain any number of Authorization Server defined payload claims. If you want those claims made available to the ABL application via a Client-Principal, set this option:

```
jwtToken.includeAllClaims={ true | false }
```

The creation and sealing of a Client-Principal is handled by the `OEClientPrincipalFilter` and follows its usual configuration rules.

Debug Logging for OAuth2

OAuth2 support can be troublesome to configure due to the various actor implementations by different vendors. When Client access is denied, you may have to troubleshoot your Spring OAuth2 configuration. It is often easier to setup a dedicated PAS for OpenEdge test instance and enable full Spring Security and OpenEdge logging. It will produce large amounts of information, which may be filtered once you gain a better understanding of the problem area.

Open the *web-app*`/WEB-INF/logging.xml` file. Un-comment the following lines, and set their value to "DEBUG":

```
<logger name="org.springframework.security.oauth2" level="DEBUG" />
<logger name="org.springframework.security.jwt" level="DEBUG" />
<logger name="org.springframework" level="DEBUG"/>
<logger name="com.progress.appserv.services.security" level="DEBUG"/>
```

Authenticate using SAML

Security Assertion Markup Language, or SAML, is an XML-based specification that defines how an identity provider (a SAML authority) and a service provider can exchange authentication and authorization data. It is developed and maintained by the Organization for the Advancement of Structured Information Standards (OASIS).

SAML is a network protocol. It stipulates that an identity provider must generate a SAML assertion containing information about an authenticated user, that is then transported over HTTP to a SAML service provider. A single SAML assertion can be used as an authentication token by many service providers across different security domains, enabling enterprises to implement cross-domain Single Sign-On (SSO).

SAML-based SSO benefits both application users and enterprises. It enables users to access multiple applications without having to log in each time. For enterprises, using an identity provider as a SAML authority gives them the ability to manage users, roles, and permissions from a central authentication and authorization repository.

Progress Application Server for OpenEdge (PAS for OpenEdge) supports the latest version of SAML--SAML 2.0. A PAS for OpenEdge web application can be configured to act as a SAML service provider, thus enabling an OpenEdge domain to participate in a cross-domain SAML SSO implementation.

Note that PAS for OpenEdge does not act as a direct-login service provider. Instead, it is the responsibility of a PAS for OpenEdge client application to accept requests from an end user and communicate with an identity provider to obtain SAML assertions for authenticated users.

Once the PAS for OpenEdge client application receives a SAML assertion from an identity provider, it needs to send the assertion to a PAS for OpenEdge web application, that then validates the assertion. If the validation is successful, PAS for OpenEdge automatically creates an ABL Client-Principal object, that is used to authenticate the user across all parts of the ABL application, including its databases, thus implementing SSO all the way through.

SAML concepts and terms

The growth in the adoption of SAML is closely linked to the proliferation of SaaS applications. As users begin to access the bulk of their applications over the web, enterprises need a way to ensure that user identities are centrally managed and delivered to application providers in a secure, standard way. The SAML specification addresses this need by identifying the roles of the parties involved in exchanging authentication data and specifying a standard way that authentication data needs to be structured and delivered.

The current version of the specification is **SAML 2.0**. The concepts described in this section are a small subset of the SAML specification; they relate to only those elements that you need to understand to implement SAML in PAS for OpenEdge.

If you want to learn more about SAML, see the official SAML technical overview.

Participants

In a typical SAML-based SSO workflow, there are three key participants:

- A **user**, also called a **principal**, who needs to access a service and whose identity needs to be authenticated.
- A **SAML service provider**, which is typically a web application that provides services to authenticated users and can process SAML assertions.
- An **identity provider**, also known as a **SAML authority**, that manages user identities and provides authentication and authorization services. Examples of identity providers include Okta, OneLogin, and Active Directory Federation Services.

In addition, a user may require a user agent, such as a browser, to access the desired services.

Workflow

At a high-level, a SAML-based SSO workflow proceeds as follows:

1. A user tries to access a resource or service hosted by the service provider.
2. The service provider creates a SAML request and redirects the user's browser to a configured SSO URL that belongs to an identity provider.
3. The identity provider processes the SAML request and presents an authentication challenge such as a login page or a two-factor/multi-factor authentication form to the user.
4. The user responds to the authentication challenge.
5. If the identity provider is able to authenticate the user, it sends a SAML response containing a SAML assertion to the user's browser and redirects the browser to an Assertion Consumer Service (ACS) URL that belongs to the service provider.
6. The service provider receives the SAML response and validates the SAML assertion. It then grants access to the requested service or resource.

This type of workflow is called Service Provider Initiated SSO. There is also another type of workflow that is less common, called Identity Provider Initiated SSO, where a user logs in to the identity provider first and then accesses a service provider page.

Protocols, bindings, and profiles

The SAML specification caters to a wide range of authentication requirements. As a result, it defines a number of protocols, bindings, and profiles, each of which pertain to specific authentication use cases.

A *protocol* is a standard way to perform an operation. For example, the SAML specification defines an Authentication Request protocol that service providers must use when they need to send authentication requests to an identity provider.

A *binding* defines how a protocol message must be carried over a transport layer. The SAML specification defines bindings for various types of HTTP as well as SOAP requests.

A *profile* is a combination of bindings and protocols. For example, the Web Browser SSO Profile defines that when a service provider needs to authenticate a user who is trying to access a resource via a browser, it should send an Authentication Request message to an identity provider using either the HTTP Redirect, HTTP POST, or HTTP Artifact binding.

SAML assertions

At the core of SAML is a SAML assertion (also called a SAML token). When an identity provider authenticates a user's identity, it generates a SAML assertion that it returns as part of a SAML response. A SAML assertion in the Web Browser SSO Profile contains the following:

- Information about the assertion itself, such as the SAML version, the timestamp when the assertion was issued, the identity provider that issued it, etc.
- Information about the user (referred to in the assertion as the **subject**). The assertion typically specifies the Name ID of the user (*joe@hiscompany.com*) and also specifies the format of the Name ID (*email address*, *Windows domain qualified name*, *Kerberos principal name*, etc).
- Validity period of the assertion; may include conditions such as not before or after a specified date and time.
- An `<AudienceRestriction>` element that specifies which URIs are valid for the assertion. The ACS URL must be listed as one of the audiences in this element.
- Optional attributes that can be customized based on your needs. A typical use of these optional attributes is pass information about the user's roles.

A SAML response can be signed or unsigned and a SAML assertion can be signed and/or encrypted. The service provider as well as the identity provider must be configured so that there is an 'agreement' on the signatures and encryptions. This is usually done by exposing configurations through metadata.

Metadata

Since digital signatures and encryption can be used in SAML messages, the service provider and the identity provider must establish trust in advance. This is typically done through metadata defined in an XML file.

The metadata of an identity provider includes information such as:

- The entity ID--a unique identifier for the identity provider. This is typically an endpoint or URL.
- Supported HTTP bindings (POST, Redirect, etc) and the SSO URL for each binding.
- The identity provider's X.509 public key certificate that is used by the service provider to decrypt SAML response messages.

The metadata of the service provider includes information such as:

- The entity ID--a unique identifier for the service provider. This is typically an endpoint or URL.
- The Assertion Consumer Service (ACS) URL. This is the location where the service provider receives SAML response messages. The ACS URL is typically sent in the SAML request message as well.
- The service provider's X.509 public key certificate that is used by the identity provider to decrypt SAML request messages.

There are a few elements whose values must match in the metadata of the identity provider and the service provider. For example:

- The Name ID format of the user whose identity needs to be authenticated.
- The hashing algorithm if digital signatures are being used.

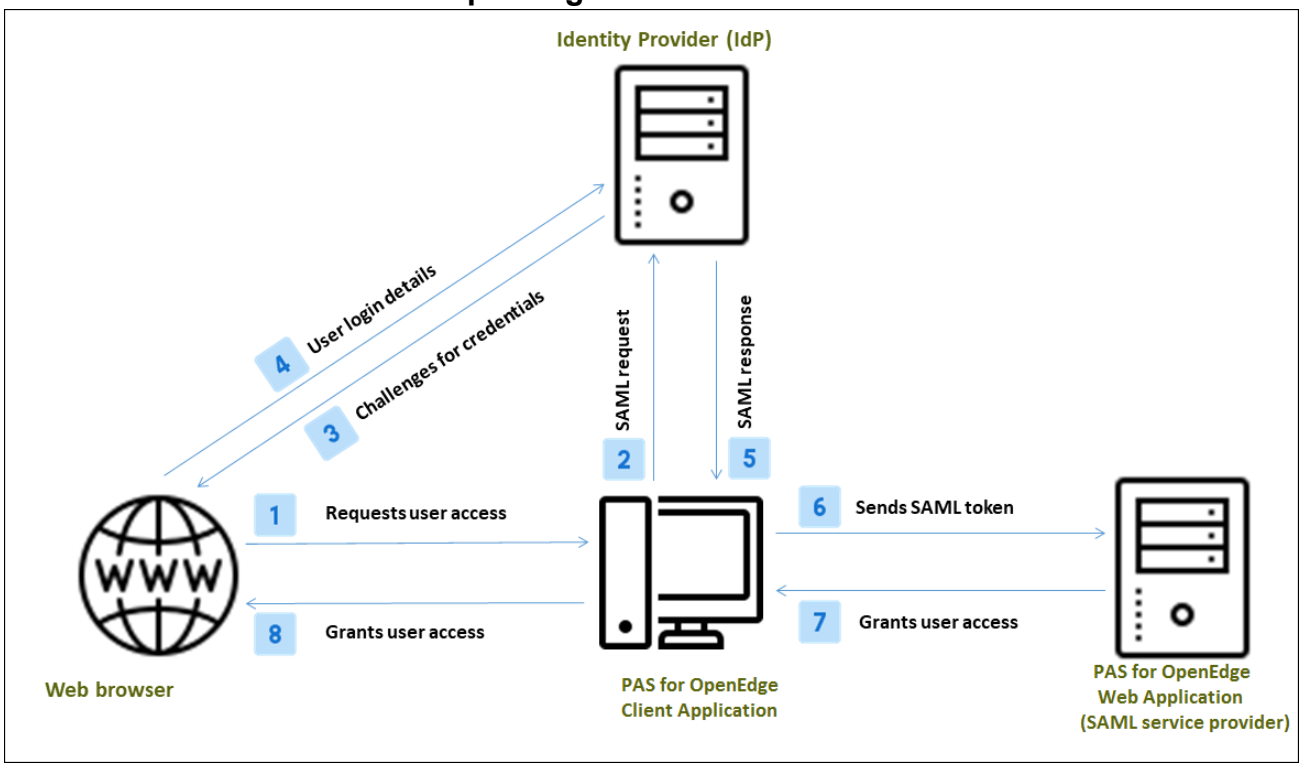
Support for SAML in PAS for OpenEdge

It is important to note that PAS for OpenEdge does not serve as a direct-login service provider. This means that end users do not directly access a PAS for OpenEdge web application. Instead, a PAS for OpenEdge client application acts as the intermediary between the user and the PAS for OpenEdge web application. A PAS for OpenEdge client application can be any REST or JavaScript program that can send and receive HTTP messages.

The client application is also responsible for sending SAML requests to an identity provider, receiving a SAML assertion from it, and then sending the assertion to the PAS for OpenEdge web application. The assertion must be sent via the basic authentication header in an HTTP request.

To set up a PAS for OpenEdge web application as a SAML service provider, you must configure the web application's `oeablSecurity.properties` file.

How SAML works in PAS for OpenEdge



A SAML workflow that involves PAS for OpenEdge is similar to any Service Provider Initiated SSO workflow, but with a few additional steps:

1. A user uses a browser to connect to a PAS for OpenEdge client application and tries to perform an operation that requires a resource deployed on a PAS for OpenEdge web application.
2. The PAS for OpenEdge client application generates a SAML authentication request and redirects the user's browser to the configured identity provider's SSO URL.
3. The identity provider processes the SAML request and presents the user with an authentication challenge.
4. The user responds to the authentication challenge by supplying the required credentials.
5. The identity provider authenticates the user and generates a SAML response containing a SAML assertion and redirects the user's browser to the client application's ACS URL.
6. The PAS for OpenEdge client application receives the SAML response, processes it, and sends the SAML assertion to the PAS for OpenEdge web application by passing it in an HTTP Authorization header.
7. The PAS for OpenEdge web application processes the SAML assertion based on properties configured in its `oeablSecurity.properties` file. It performs tasks such as:
   a. **Decryption**: If the assertion is encrypted and/or signed, the PAS for OpenEdge web application uses the identity provider's public key (exposed via the provider's metadata) to decrypt the message.
   b. **Validity Period Check**: The PAS for OpenEdge web application verifies that the assertion is not past its expiry date.
   c. **First Level HTTP Method Validation**: The PAS for OpenEdge web application checks if the HTTP method used by the PAS client to send the SAML assertion is a permitted method.
   d. **ACS URL Validation (Optional)**: Depending on the property settings, the PAS for OpenEdge web application may also match the assertion's `<AudienceRestriction>` element against the PAS client's ACS URL (exposed via the client's metadata).
   e. **Authorization**: The PAS for OpenEdge web application obtains the user's roles from attribute statements in the SAML assertion and determines if the user is authorized to access the requested resource. It does this by matching the roles obtained from the assertion with the roles defined for resource URLs in the `oeablSecurity.csv` file.
   f. **Second Level HTTP Method Validation**: The PAS for OpenEdge web application verifies that the HTTP method used in the request matches the permitted HTTP method defined for the requested resource URL in the `oeablSecurity.csv` file.
   g. **Client-Principal Creation**: If the PAS for OpenEdge web application is able to perform the previous tasks successfully, it creates a sealed Client-Principal object and passes the object to the relevant ABL resource.
   h. **Response**: Finally, the PAS for OpenEdge web application sends a response to the client application.
8. The client application processes the response and grants access to the end user to perform their desired operation.

How SAML assertion elements map to Client-Principal object attributes

If the PAS for OpenEdge web application is able to validate the SAML assertion, it creates a sealed Client-Principal object and passes it to the required ABL resource.

The following attributes of the resulting Client-Principal object are populated from the SAML assertion's fields:

Client-Principal attribute	SAML assertion field
USER-ID[1]	NameID[1]
DOMAIN-NAME[1]	NameID[1]
SESSION-ID	SessionIndex attribute in AuthnStatement
LOGIN-EXPIRATION-TIMESTAMP[3]	NotOnOrAfter attribute in Conditions
ROLES[2]	Optional attributes for roles [2]

1 If the NameID is an email address, PAS for OpenEdge uses the '@' separator to extract the user-id and the domain name.

2 The names of these optional attributes must be set in the samlToken.UserDetails.roleAttrName property in the oeablSecurity.properties file.

3 The oeablSecurity.properties file has a property named samlToken.webSSOProcessingFilter.responseSkew. If a value is specified for this property (in seconds), it is added to the NotOnOrAfter timestamp.

Other attribute values are populated based on the OEClientPrincipalFilter properties set in the oeablSecurity.properties file.

If there are other attributes that you want to set, that are not covered by either the SAML assertion or the default OEClientPrincipalFilter, you can set them using any of the 10 key-value OEClientPrincipalFilter static properties. For example, if you want to set the DOMAIN-DESCRIPTION attribute, you can modify a key-value property as follows:

```
OEClientPrincipalFilter.properties.1.key=DOMAIN-DESCRIPTION
 OEClientPrincipalFilter.properties.1.value=OEApplication
```

Implementing SAML in a PAS for OpenEdge web application

To implement SAML in PAS for OpenEdge, you need to perform two tasks:

- Set up a PAS for OpenEdge client application.
- Configure SAML properties in a PAS for OpenEdge web application.

Setting up a PAS for OpenEdge client application

Since a PAS for OpenEdge web application does not act as a direct-login service provider, you need to develop a PAS for OpenEdge client application that can communicate with end users and work with identity providers.

The common use case is to develop a client application (for example, a Kendo UI Builder application) that provides HTML pages that end users access via a browser. When the end user clicks a link or URL that requires a protected OpenEdge resource, the PAS for OpenEdge client initiates a SAML workflow. This resource could be, for example, an ABL business entity wrapped in a Data Object Service, in which case you would want your PAS client to call a URL such as http://*host*:*port*/*OpenEdgeApplication*/rest/*DataObjService*/*ABLBusinessEntity*.

To implement this use case, you must develop your PAS client application and set it up to work with an identity provider of your choice. The important thing to remember is that the PAS client application must send the SAML assertion that it receives from an identity provider to the PAS for OpenEdge web application.

Note:

To obtain a protected resource from PAS for OpenEdge, the client application must send a valid SAML assertion to the PAS for OpenEdge web application. This assertion *must* be sent in the **Authorization header** of the HTTP request with the prefix **SAML**.

Also, the HTTP method that your PAS client uses to send SAML assertions must match the `samlToken.httpBinding.allowedMethods` property set in the `oeablSecurity.properties` file. It must also match a method defined for the resource URL in the `oeablSecurity.csv` file.

In addition, you need to generate a metadata file for your PAS client application. This task is performed either by an application developer or a system administrator. There are tools available online (such as this one) that can help generate this metadata. The metadata is useful not only when you are configuring the identity provider to communicate with your PAS for OpenEdge client application, but also when you are configuring SAML properties in the PAS for OpenEdge web application.

Configuring SAML properties in a PAS for OpenEdge web application

After you have set up your client application, configure the following properties in your PAS for OpenEdge web application's `oeablSecurity.properties` file located in the `WEB-INF` folder.

Note: Before you configure these properties you should download the metadata file of the client application as well as the identity provider and store them in a location that is accessible to your PAS for OpenEdge web application.

Client login model

Set the value of the `client.login.model` property to `saml`.

```
client.login.model=saml
```

Metadata

Set the path to the PAS client application's (service provider's) metadata.

```
samlToken.metadata.spMetaDataFileLocation=<path to PAS client application's metadata>
```

Set the path to the identity provider's metadata.

```
samlToken.metadata.idpMetaDataFileLocation=<path to IdP's metadata>
```

HTTP Method Validation

Set the allowed HTTP methods for incoming HTTP requests that contain the SAML assertion. If the only allowed method is `GET`, and the SAML assertion is part of a `POST` message, PAS for OpenEdge rejects the request.

```
samlToken.httpBinding.allowMethods=GET,POST,PUT,DELETE
```

ACS URL Validation

The `Audience restriction` URL, a field in the SAML assertion that Identity Provider returns, must match the location value of the `AssertionConsumerService` URL in the Service Provider's metadata file. If not, the Service Provider rejects the SAML response from the Identity Provider. This property is optional and the URL matching can be skipped. However, if you need to check ACS endpoint, set this property to `true`.

```
samlToken.webSSOProcessingFilter.checkForACSEndpointUrl={false|true}
```

Validity period

PAS for OpenEdge rejects a request if the SAML assertion's `NotOnOrAfter` timestamp has passed. However, you can set the following property to extend the duration of the assertion's validity period.

```
samlToken.webSSOProcessingFilter.responseSkew=<time in seconds>
```

Roles

If the incoming SAML assertion's attribute statements define user roles, you can set the attribute names as comma-separated values in the following property. This enables PAS for OpenEdge to populate the `ROLES` attribute when creating the Client-Principal object.

```
samlToken.UserDetails.roleAttrName=Attribute1,Attribute2
```

Note that PAS for OpenEdge authorizes a user to access a resource only if the role values in the SAML assertion's attributes map to roles defined for resource URLs in the `oeablSecurity.csv` file. By default, these roles have a `ROLE_` prefix. If the incoming role values do not contain the prefix, you an use the following property to add it:

```
samlToken.UserDetails.rolePrefix=ROLE_
```

Domains

If the NameID passed in the SAML assertion is an email address (for example *johndoe@hisdomain.com*), PAS for OpenEdge extracts the domain name using the '`@`' separator and assigns it to the `DOMAIN-NAME` attribute of the Client-Principal object that it creates. However, if the NameID is not an email address, you can use the following property to set a default domain name:

```
samlToken.UserDetails.userDomain=Domain_Name
```

You may want to specify a domain name that has minimal access rights to your ABL application.

Note: If `saml` is set as the client login model, this property overrides the `OEClientPrincipalFilter.domain` property.

To create a sealed Client-Principal object, PAS for OpenEdge requires a domain access code that corresponds to the domain name. Typically, a CSV file is created containing a list of key-value pairs (for example, *domain, domain-access-code*) and then encrypted into a Java keystore file. To specify this keystore file, use the following property:

```
samlToken.UserDetails.registryFile=keystore_file
```

Note: If `saml` is set as the client login model, this property overrides the `OEClientPrincipalFilter.registryFile` property. To learn more about creating registry files, see Configuring multiple domains.

Debug logging for SAML

Due to various role implementations by different vendors, you may have to troubleshoot your SAML configuration when a problem occurs. You can enable OpenEdge logging and use the required information to debug the problem.

Open the *web-app*/WEB-INF/logging.xml file. Un-comment the following lines, and set their value to "DEBUG":

```
<logger name="org.springframework.security.saml" level="DEBUG"/>
<logger name="org.opensaml" level="DEBUG"/>
<logger name=" PROTOCOL_MESSAGE " level="DEBUG"/>
```

9

WebSpeed configuration and management

PAS for OpenEdge is available as a host for WebSpeed applications. You can migrate existing WebSpeed applications to PAS for OpenEdge, or you can use the Progress Developer Studio for OpenEdge to develop new WebSpeed applications that run on PAS for OpenEdge.

Progress WebSpeed is a Progress ABL development and deployment environment. WebSpeed allows you to build applications that use HTML, XML, WML, DHTML, and most other mark-up languages (MLs) as the user interface. This means that WebSpeed can be used for applications where users access the application using a web browser, a mobile phone, or some other system making requests for information using XML and HTTP or HTTP/S as the transport protocol. In addition to building Web-based applications by design, you can use WebSpeed to web-enable existing Progress OpenEdge applications that previously ran stand-alone or as an Progress AppServer-based application.

With WebSpeed, you can develop and deploy:

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

As a PAS for OpenEdge system administrators, you should know how to:

- Deploy a classic WebSpeed application to a PAS for OpenEdge instance.
- Deploy a new WebHandler WebSpeed application to a PAS for OpenEdge instance.

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

Note: Also see Deploy a new WebHandler Webspeed application to a PAS for OpenEdge instance on page 21 for information about changing the default transport that handles WebSpeed requests.

For details, see the following topics:

- Tools for managing and configuring WebSpeed on PAS for OpenEdge
- Deploy a classic Webspeed application to a PAS for OpenEdge instance
- Secure the Web transport
- Migrating classic WebSpeed applications

Tools for managing and configuring WebSpeed on PAS for OpenEdge

This section is an overview of the tools for configuring and managing WebSpeed on PAS for OpenEdge.

Note: WebTools, a collection of browser-based utilities implemented for classic WebSpeed, are not supported for use with WebSpeed on PAS for OpenEdge.

Command line utilities

The command line utilities for classic WebSpeed (WTBMAN and WSCONFIG) are not applicable to WebSpeed on PAS for OpenEdge. Instead, you use the standard PAS for OpenEdge utilities, namely:

- TCMAN — Similar to WTBMAN, it is used to control and monitor the server. TCMAN has additional functionality, compared to WTBMAN, for creating and configuring server instances, managing applications, and more.
- OEPROP — Similar to MERGEPROP. which is used to manage the `ubroker.properties` file for classic WebSpeed. Use OEPROP to manage the `openedge.properties` file, which is similar in form and function to `ubroker.properties`.

REST APIs

The `oemanager.war` Java Web application provides a REST API for remote management and monitoring of sessions, agents, transports, and of ABL applications deployed on a Progress Application Server for OpenEdge instance. Functionality is implemented as a REST service that is accessed via a URI.

OEM

You can configure and manage both classic WebSpeed and WebSpeed on PAS for OpenEdge from the OEM tool. For more information, see the online help in OEM. For information specific to managing WebSpeed on PAS for OpenEdge, see the *OpenEdge Management: Progress Application Server for OpenEdge Configuration* book.

Manually editing configuration files

You can edit `openedge.properties` (and other PAS for OpenEdge properties files) with a text editor. The properties are documented in the `/conf/openedge.properties.README` file.

Note: Be aware that manual edits cannot be tracked. Therefore, if manual changes are made, and changes are also made with TCMAN, OEPROP, REST APIs, or OEM tool, the manual changes may be lost.

See also

TCMAN Reference on page 199
OEPROP on page 50
REST API Reference for oemanager.war on page 243

Deploy a classic Webspeed application to a PAS for OpenEdge instance

In classic WebSpeed, you configure the WebSpeed Transaction Server and define features with environment variables and entries in the `ubroker.properties` file. In PAS for OpenEdge, the `$CATALINA_BASE/conf/openedge.properties` file is used instead of `ubroker.properties` largely replaces environment variables. It is similar in format to `ubroker.properties` but applies only to a single server instance. That single instance, however, can support a number of connection and session types.

For example, you specify database connections for all deployed Web applications (including WebSpeed) in `openedge.properties` by setting `agentStartupParam` parameter with the following syntax:

```
agentStartupParam=-db dbname [ -db dbname... ]
```

In addition, `openedge.properties` contains a `ROOT.WEB` section the applies specifically to WebSpeed support. For example:

```
[oepas1.ROOT.WEB]
    adapterEnabled=1
    defaultCookieDomain=
    defaultCookiePath=
    defaultHandler=OpenEdge.Web.CompatibilityHandler
    srvrDebug=0
```

You can find help that explains these parameters and their settings in `$CATALINA_BASE/conf/openedge.properties.README`

Migrating a WebSpeed application to a PAS for OpenEdge instance, involves moving the application's static files to a specific folder in the instance, and updating the instance's PROPATH to include the folders that contain the application's r-code.

Static files

A PAS for OpenEdge instance expects the static files that support a WebSpeed application to be in a particular location in the instance's directory structure. Static files include images and html files.

The location of static files for the default Web application is:

```
instance_name/webapps/ROOT/static
```

If you deploy another application, the default location for its static files is:

```
instance_name/webapps/webapp_name/static
```

where *webapp_name* is the name of the WebSpeed application.

r-code

To enable a PAS for OpenEdge instance to find a WebSpeed application's r-code, add any folders that contain WebSpeed application r-code files to the instance agent's PROPATH.

The default location for r-code is:

```
instance_name/openedge
```

PROPATH is set in the instance's `../conf/openedge.properties` file. For example:

```
             .
             .
             .
[AppServer.Agent]
 agentMaxPort=62202
 agentMinPort=62002
 agentShutdownProc=
 agentStartupProc=
 agentStartupProcParam=
 collectStatsData=0
 flushStatsData=0
 infoVersion=9010
 keyAlias=
 keyAliasPasswd=
 keyStorePasswd=
 keyStorePath=.\keys\
 lockAllExtLib=
 lockAllNonThreadSafeExtLib=
 noSessionCache=0
 numInitialSessions=5
 PROPATH=${CATALINA_BASE}/openedge,${DLC}/tty,${DLC}/tty/netlib/OpenEdge.Net.pl
             .
             .
             .
```

Note: `CATALINA_BASE` is an environment variable that resolves to *instance_path*.

Secure the Web transport

PASOE web transport's default security is set very coarse grained, meaning that if a client is authenticated and has the right role they can access anything in the /web/** relative URI.

There are two possible solutions to adding additional fine grained access control:

- Write ABL code in your web handlers that authorizes the user (Client-Principal) to a particular handler's URI or to sub paths
- Add Spring Security <intercept-url> elements in your oeablSecurity-*.xml configuration - with one <intercept-url> per web handler

The former exposes the ABL business logic to additional risk and requires yet another location where role based authorization has to be configured and implemented. The later is safer (because it runs in protected code space) and is easily configurable at the end production site.

Migrating classic WebSpeed applications

Migrating a WebSpeed application to a PAS for OpenEdge instance, involves moving the application's static files to a specific folder in the instance, and updating the instance's PROPATH to include the folders that contain the application's r-code.

Static files

A PAS for OpenEdge instance expects the static files that support a WebSpeed application to be in a particular location in the instance's directory structure. Static files include images and html files.

The location of static files for the default Web application is:

```
instance_name/webapps/ROOT/static
```

If you deploy another application, the default location for its static files is:

```
instance_name/webapps/webapp_name/static
```

where *webapp_name* is the name of the WebSpeed application.

r-code

To enable a PAS for OpenEdge instance to find a WebSpeed application's r-code, add any folders that contain WebSpeed application r-code files to the instance agent's PROPATH.

The default location for r-code is:

```
instance_name/openedge
```

PROPATH is set in the instance's `../conf/openedge.properties` file. For example:

```
             .
             .
             .
[AppServer.Agent]
 agentMaxPort=62202
 agentMinPort=62002
 agentShutdownProc=
 agentStartupProc=
 agentStartupProcParam=
 collectStatsData=0
 flushStatsData=0
 infoVersion=9010
 keyAlias=
 keyAliasPasswd=
 keyStorePasswd=
 keyStorePath=.\keys\
 lockAllExtLib=
 lockAllNonThreadSafeExtLib=
 noSessionCache=0
 numInitialSessions=5
 PROPATH=${CATALINA_BASE}/openedge,${DLC}/tty,${DLC}/tty/netlib/OpenEdge.Net.pl
             .
             .
             .
```

Note: `CATALINA_BASE` is an environment variable that resolves to *`instance_path`*.

10

TCMAN Reference

TCMAN is a command-line utility for managing and administering the Progress Application Server. TCMAN extends the basic Tomcat scripts for starting, stopping, and managing server instances.

This *TCMAN Reference* contains usage information for the `tcman` command as well as syntax information on all of the TCMAN actions.

For details, see the following topics:

- Using TCMAN
- Manager actions
- Server actions
- General actions

Using TCMAN

You use the TCMAN utility by running the `tcman` command, with specified actions and options, from either the core PAS (`$CATALINA_HOME/bin`) or from an instance (`$CATALINA_BASE/bin`). You can run `tcman` from a command line, or in a script. Both Windows (`tcman.bat`) and UNIX (`tcman.sh`) versions of TCMAN are implemented in the Progress Application Server.

See also

The tcman command on page 200
Extending TCMAN on page 202

Manager actions on page 204
Server actions on page 216
General actions on page 239

The tcman command

Purpose

TCMAN is a command-line utility for managing and administering PAS. On UNIX systems, you run the `tcman.sh` script followed by appropriate TCMAN actions and options. On Windows systems, you run the `tcman.bat` batch file, which is identical syntactically and functionally with `tcman.sh`.

Note: For the sake of brevity, all the syntax statements and examples in this reference show the `tcman.sh` script.

Syntax

```
{ $CATALINA_HOME | $CATALINA_BASE } /bin/tcman.sh action  [ general_options ]
[ action_options ]
```

Parameters

`$CATALINA_HOME|$CATALINA_BASE`

Specify whether to run TCMAN from the root directory of the installed PAS (`$CATALINA_HOME`) or from the root directory of an instance (`$CATALINA_BASE`). The context of where you run TCMAN (whether from the `/bin` directory of the parent, or the `/bin` directory of an instance) affects which server the utility acts on.

Note: TCMAN automatically determines the value of `CATALINA_BASE` from the directory where you start it. When you run it from the `/bin` directory of an instance, the value of `CATALINA_BASE` is the root directory of the instance. If you run it from the `/bin` directory of the installed Progress Application Server, the value of `CATALINA_BASE` is the root directory of the installed server (which is the same value as `CATALINA_HOME`).

`action`

Specify which TCMAN action to invoke.

`general_options`

Specify one or more of the TCMAN common options that can apply to most actions. Note that one or more of the general options may be required by a specific action. For example, the `list` action requires `-u` in order to pass a user name and password.

The output of `tcman.sh help action` includes a list of general options that are applicable to a particular action.

The following table is a list of the common options:

Table 16: TCMAN general options

Common options	Function
-u *user_name*:*password*	Pass a valid user name and a password for HTTP Basic access authentication.
-v	Display verbose output.
-M *URL*	Override the default manager that manages Web applications by specifying the URL of an alternative manager. *URL* is expressed in the following format: {http|https}://*host*:*port*/*manager_application*
-B	Override default CATALINA_BASE environment settings.
-n	Debug the TCMAN action but do not execute changes.
-I *instance_name*	Run TCMAN from the /bin directory of the specified instance.

action_options

Specify an option that applies to the selected *action*. These options are explained in the topics that describe each action.

Example

Run the `help` action from the core server (`/psc/pashome`) to display a list of available TCMAN actions:

```
/psc/pashome/bin/tcman.sh help
usage:  tcman action [options...]
  manager actions:
      list         list deployed applications
      info         list server info
      deploy       deploy application
      undeploy     undeploy application
      reload       reload application
      status       show server status
      leaks        show server memory leaks
      enable       start web application running
      disable      stop running web application
      resources    list server global resources
      sessions     list a web application's sessions
  server actions:
      create       create a new server instance
      delete       delete server instance
      config       dump CATALINA_BASE configuration
      clean        clean/archive log files
      instances    list tracked server instances
      register     manually register an instance
      unregister   manually unregister an instance
      start        start this server
      stop         stop this server
      version      show the server version information
      test         test the server's configuration
  general actions:
      env          show tcman execution environment
      help         show this information
```

See also

Manager actions on page 204
Server actions on page 216
Extending TCMAN on page 202
General actions on page 239

Extending TCMAN

You can extend TCMAN by adding your own actions to the built-in actions of the TCMAN utility. You implement actions as UNIX shell scripts, Windows batch files, or Windows PowerShell scripts. Customizing TCMAN with your own actions allows you to unify all of your administrative tasks under a single utility.

File name format

Name your TCMAN action scripts using the following convention:

*action_name*Cmd{.sh | .bat | .ps1}

where file name extensions `.sh`, `.bat`, and `.ps1` are used for UNIX scripts, Windows batch files, and PowerShell scripts respectively.

UNIX action scripts

UNIX action scripts must be executable. That is, the file must have execution permissions for the current user. Each script must exit with a zero (0) status when successful, or a positive non-zero value to indicate an error.

When the action script is in the core server's bin directory (`$CATALINA_HOME/bin`), it is available to administrators of any instance. When the action script is in an instance's bin directory (`$CATALINA_BASE/bin`), it is only available to administrators of that instance.

The search order is:

1. `$CATALINA_BASE/bin`
2. `$CATALINA_HOME/bin`

Windows batch and PowerShell action scripts

On Windows, action scripts can be either DOS batch scripts (`.bat`) or PowerShell scripts (`.ps1`). Batch scripts should return zero (0) for success, or a positive non-zero integer to indicate an error level. Powershell scripts should return `\$true` for success or `\$false` to indicate an error.

When the action script is in the core server's bin directory (`$CATALINA_HOME/bin`), it is available to administrators of any instance. When the action script is in an instance's bin directory (`$CATALINA_BASE/bin`), it is only available to administrators of that instance.

If you have both batch and PowerShell scripts, TCMAN gives precedence to batch files. The search order is:

1. `$CATALINA_BASE/bin/`*`action_name`*`Cmd.bat`
2. `$CATALINA_BASE/bin/`*`action_name`*`Cmd.ps1`
3. `$CATALINA_HOME/bin/`*`action_name`*`Cmd.bat`
4. `$CATALINA_HOME/bin/`*`action_name`*`Cmd.ps1`

Command-line Syntax

You invoke an action script by specifying the *`action-name`* segment of a script's file name in a TCMAN command line.

```
tcman.sh action-name [general_options] [ action-name_options]
```

Note: You do not include the `Cmd.`*`extension`* portion of the file name on the TCMAN command line.

TCMAN first looks for action scripts in the instance (`$CATALINA_BASE/bin`) and then in the core server (`$CATALINA_HOME/bin`). When it finds the action script, TCMAN passes all command line options and parameters during execution.

Example

The following command line runs a script file named `viewCmd.sh` and passes the `-v` option and the `logs/catalina.out` argument.

```
tcman.sh view -v  logs/catalina.out
```

Notes

- When TCMAN does not recognize *action-name* as one of the built-in actions, it looks for an action script . If the specified action is not a built-in action or external action script it will exit with an error.
- When the external action script exits, TCMAN uses its return status to either issue an error or to exit with success. Therefore, extension action scripts should always return a specific value according to the platform type.
- Before executing an external action script, TCMAN exports these environment variables for consumption by the action script file:
  - JAVA_HOME : the root directory of the JAVA JDK (optional)
  - JRE_HOME : the root directory of the JAVA JRE (optional)
  - CATALINA_HOME : the root directory of where PAS was installed
  - CATALINA_BASE : the root directory of the PAS instance being managed (can be the same as CATALINA_HOME)
  - CATALINA_TMPDIR : the temporary file directory for temp files
  - CATALINA_PID : the file where the PAS process-id is stored
  - tcman_osshell : The name of the OS and command line shell TCMAN is running in
  - tcman_instancebase : the name of the new instance base directory when executing a create action
  - tcman_httpport : the HTTP network port command line option (`-p`)
  - tcman_httpsport : the HTTPS network port command line option (`-P`)
  - tcman_cfghttpport : the current instance's configured HTTP network port
  - tcman_cfghttpsport : the current instance's configuredHTTPS network port
  - tcman_cfgshutport : the current instance's shutdown network port (can be `-1` if there is no network shutdown port)
  - tcman_alias : the instance's alias name
  - tcman_type : the type of instance (home, service (on Windows), instance)
  - tcman_secmodel : the security model to apply (`default`, `production`, or `development`)
  - tcman_umask : the UNIX umask to use when creating new directories and files
  - tcman_verbose : the state of the TCMAN `-v` command line option
  - tcman_debug : The state of the TCMAN `-g` command line option

Manager actions

This section details the actions available for deploying, running, and monitoring web applications on a server instance.

See also

The tcman command on page 200

List deployed applications (list)

Purpose

Display all the web applications that are deployed on an instance.

Note: This command may be used whether the instance is online or offline. However, the output differs. When used offline, TCMAN simply shows a list of deployed application directories in the instance's web applications directory. When used online, it provides additional run-time details about the deployed web applications.

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance if the instance is online. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

Syntax

```
tcman.sh list [general_options] [-u user_id:password]
```

Parameters

general_options

Specify one or more of the options that can be used with any TCMAN action. Run `tcman.sh help list` to see which general options are appropriate.

`-u` *user_id*`:`*password*

Specify a valid user name and password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

Note: This option is required if the server is online. It is not required if the server is offline.

Example

Show the Web applications deployed to `acme1` when the instance is online:

```
/psc/acme1/bin/tcman.sh list -u tomcat:tomcat
OK - Listed applications for virtual host localhost
/:running:0:ROOT
/manager:running:4:manager
/oemanager:running:0:oemanager
/oeadapters:running:0:oeabl
```

Show the Web applications deployed to `acme1` when the instance is offline:

```
 /psc/acme1/bin/tcman.sh list
OK - Listing directories for /psc/acme1/webapps
/manager:stopped:0:manager
/oeadapters:stopped:0:oeabl
/oemanager:stopped:0:oemanager
/:stopped:0:ROOT
```

See also

Display Web application HTTP sessions (sessions) on page 216
Deploy a Web application (deploy) on page 207
Undeploy a Web application (undeploy) on page 209
Start a Web application (enable) on page 213
Stop a Web application (disable) on page 214
The tcman command on page 200

Display OS and server information (info)

Purpose

Display server and OS information for a running instance.

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

Use the `test` action to show configuration information about a server that is not running.

Syntax

```
tcman.sh info [general_options]  -u user_name:password
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help info` to see which general options are appropriate.

`-u` *user_name*`:`*password*

Pass a valid user name and a password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

Example

Display the OS and server information for the running instance named `acme1`:

```
$: /psc/pashome/tcman.sh info -I acme1 -u tomcat:tomcat
OK - Server info
Tomcat Version: Apache Tomcat/7.0.42
OS Name: Linux
OS Version: 2.6.18-164.el5
OS Architecture: amd64
JVM Version: 1.7.0_02-b13
JVM Vendor: Oracle Corporation
```

See also

Display server, OS, and runtime version information (version) on page 234
Display detailed server status (status) on page 211
Test a server configuration (test) on page 235
The tcman command on page 200
Display or modify the server features of an instance (feature) on page 222
The tcman command on page 200

Deploy a Web application (deploy)

Purpose

Deploy an ABL web application (`.war` file) to a PAS for OpenEdge instance, online (when the server is running) or offline (when the server is not running). TCMAN copies the web application to the server's web application directory.

Syntax

```
tcman.sh deploy [general_options] -l [-u user_id:password] [-a app_name]
war_file_path [abl_app_name]
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help deploy` to see which general options are appropriate.

`-l`

Use this switch to deploy an ABL web application online without having to restart the instance. The switch loads the context necessary for the successful online deployment of a web application.

Note: The Tomcat manager (oemanager.war) must already be deployed for the `-l` switch to work. In the PAS for OpenEdge development server, the Tomcat manager is by default deployed in the default instance, `oepas1`. For all new instances in both the development and the production servers, you have to deploy the Tomcat manager first before you can use this switch. Also note that the `-l` switch can be used only in conjunction with the `-u` parameter.

`-u` *`user_id`*`:`*`password`*

Specify a valid user name and password for HTTP Basic access authentication.

Note: This parameter is required if the server is online. It is not required if the server is offline.

`-a` *`app _name`*

Specify a name for the web application. If you do not use this option, the application name will be the same as the `.war` file name.

`war_file_path`

Specify the location of the web application `.war` file that you want to deploy.

`abl_app_name`

It is the name of the ABL application under which you want to deploy your ABL web application.

An ABL application is a logical grouping of ABL web applications. It consists of a session manager, one or more multi-session agents, and one or more deployed ABL web applications.

An instance can contain more than one ABL application. You can deploy an ABL web application to either the default ABL application, which is named after the instance, or to a new ABL application. If you want to deploy an ABL web application under a new ABL application, then specify a name for the ABL application here.

Example

Deploy and rename `oeabl.war` (a web application that implements OpenEdge adapters) to the `acme1` instance of the core `pashome` server online:

```
/psc/acme1/bin/tcman.sh deploy -l -u tomcat:tomcat -a oeadapters
/psc/pashome/extras/oeabl.war
OK - deployed /psc/pashome/extras/oeabl.war to local directory /psc/acme1/webapps
```

Note: The `$CATALINA_HOME/extras` directory (`/psc/pashome/extras` in the example above) also contains number of instance management applications, including `host-manager.war`, `manager.war`, and `oemanager.war`.

See also

List deployed applications (list) on page 205
Undeploy a Web application (undeploy) on page 209
Reload a Web application (reload) on page 210
Start a Web application (enable) on page 213

Stop a Web application (disable) on page 214
The tcman command on page 200

Undeploy a Web application (undeploy)

Purpose

Remove a Web application from running (online) or stopped (offline) instances. If the instance's autodeploy option is off, you must stop and restart a running server to complete removal. Note that the autodeploy option is set in the `.../conf/appserver.properties` file and is off by default.

Syntax

```
tcman.sh undeploy [general_options] [-u user_id:password]  app _name
```

Parameters

general_options

Specify one or more of the options that can be used with any TCMAN action. Run `tcman.sh help undeploy` to see which general options are appropriate.

`-u` *user_id*`:`*password*

Specify a valid user name and password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.) This option is required if you are accessing an online instance.

app_name

Specify the name of the web application to remove.

Example

Remove the oemanager application from the `acme1` instance:

```
/psc/acme1/bin/tcman.sh undeploy -u tomcat:tomcat oemanager
              OK - Undeployed application at context path /oemanager
```

See also

Deploy a Web application (deploy) on page 207
List deployed applications (list) on page 205
Reload a Web application (reload) on page 210
The tcman command on page 200

Reload a Web application (reload)

Purpose

Restart a deployed, running Web application so that the application can pick up changes to its classes or libraries.

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

Note: The reload action does not reload the web application's `web.xml` file. To begin using changes to `web.xml`, you must stop and restart the web application.

Syntax

```
tcman.sh reload [general_options] -u user_id:password app_name
```

Parameters

general_options

Specify one or more of the options that can be used with any TCMAN action. Run `tcman.sh help reload` to see which general options are appropriate.

`-u` *user_id*`:`*password*

Specify a valid user name and password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

Note: This option is required if the server in online. It is not required if the server is offline.

app_name

Specify the name of the web application to restart.

Example

Reload the `oemanager` web application running on the `acme1` instance:

```
 /psc/acme1/bin tcman.sh reload  -u tomcat:tomcat oemanager
OK - Reloaded application at context path /oemanager
```

See also

Start a Web application (enable) on page 213
Stop a Web application (disable) on page 214
List deployed applications (list) on page 205
The tcman command on page 200

Display detailed server status (status)

Purpose

List information from the core server's memory, including web application statistics. Information includes memory pool usage, connector thread status, and connector status. Output is in XML format. (Note that redirecting the output to an XML viewer makes it more readable.)

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

Syntax

```
tcman.sh status [general_options]  -u user_name:password [-f]
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help status` to see which general options are appropriate.

`-u` *user_name*`:`*password*

Pass a valid user name and a password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

`-f`

Return full status information.

Example

Display core server's memory and web application statistics and use `xmllint` to format for readability:

```
$: tcman.sh status -u tomcat:tomcat | xmllint --format -
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="/manager/xform.xsl" ?>
<status>
  <jvm>
    <memory free="453196832" total="520028160" max="1051394048"/>
    <memorypool name="PS Eden Space" type="Heap memory" usageInit="50331648"
usageCommitted="48758784" usageMax="55967744" usageUsed="1525560"/>
    <memorypool name="PS Old Gen" type="Heap memory" usageInit="469762048"
usageCommitted="469762048" usageMax="1006632960" usageUsed="63861584"/>
    <memorypool name="PS Survivor Space" type="Heap memory" usageInit="8388608"
usageCommitted="1507328" usageMax="1507328" usageUsed="1444184"/>
    <memorypool name="Code Cache" type="Non-heap memory" usageInit="2555904"
usageCommitted="3407872" usageMax="50331648" usageUsed="3303104"/>
    <memorypool name="PS Perm Gen" type="Non-heap memory" usageInit="67108864"
usageCommitted="67108864" usageMax="134217728" usageUsed="47406400"/>
  </jvm>
  <connector name="&amp;quot;http-bio-8601&amp;quot;">
    <threadInfo maxThreads="150" currentThreadCount="0" currentThreadsBusy="0"/>
    <requestInfo maxTime="0" processingTime="0" requestCount="0" errorCount="0"
bytesReceived="0" bytesSent="0"/>
    <workers/>
  </connector>
  <connector name="&amp;quot;http-bio-8501&amp;quot;">
    <threadInfo maxThreads="300" currentThreadCount="10" currentThreadsBusy="1"/>
    <requestInfo maxTime="2008" processingTime="2116" requestCount="10" errorCount="0"
 bytesReceived="0" bytesSent="5838"/>
    <workers>
      <worker stage="S" requestProcessingTime="2" requestBytesSent="0"
requestBytesReceived="0" remoteAddr="127.0.0.1" virtualHost="localhost" method="GET"
currentUri="/manager/status" currentQueryString="XML=true" protocol="HTTP/1.1"/>
    </workers>
  </connector>
</status>
```

See also

Display OS and server information (info) on page 206
Display memory leaks (leaks) on page 212
The tcman command on page 200

Display memory leaks (leaks)

Purpose

List Web applications with potential memory leaks.

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

Syntax

```
tcman.sh leaks [general_options]  -u user_name:password
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help leaks` to see which general options are appropriate.

`-u` *user_name*`:`*password*

Pass a valid user name and a password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

Example

Display memory leaks for web applications deployed on the `acme1` server instance:

```
/psc/acme1/bin/tcman.sh leaks -u tomcat:tomcat
OK - Found potential memory leaks in the following applications:
/warehouse
```

See also

List deployed applications (list) on page 205
The tcman command on page 200

Start a Web application (enable)

Purpose

Start a web application that is deployed but not running.

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

Syntax

```
tcman.sh enable [general_options] -u user_id:password app_name
```

Parameters

general_options

Specify one or more of the options that can be used with any TCMAN action. Run `tcman.sh help start` to see which general options are appropriate.

`-u` *user_id*`:`*password*

Specify a valid user name and password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

app_name

Specify the name of the web application to start.

Note: To start the ROOT web application, you can specify / or ROOT.

Example

Start the oeabl application deployed on the acme1 instance:

```
tcman.sh enable -u tomcat:tomcat oeabl
OK - Started application at context path /oeabl
```

See also

Stop a Web application (disable) on page 214
The tcman command on page 200
The tcman command on page 200

Stop a Web application (disable)

Purpose

Stop a running Web application.

To use this action, the Tomcat manager (manager.war) must be deployed on the instance and the instance must be running. You can deploy manager.war from $CATALINA_HOME/extras.

Syntax

```
tcman.sh disable [general_options] [-u user_id:password] app_name
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run tcman.sh help disable to see which general options are appropriate.

-u *user_id*:*password*

Specify a valid user name and password for HTTP Basic access authentication. (The default is -u tomcat:tomcat.)

app_name

Specify the name of the web application to disable.

Note: To disable the ROOT web application, you can specify / or ROOT.

Example title

Disable the oeabl application running on the acme1 instance:

```
/psc/acme1/bin/tcman.sh disable -u tomcat:tomcat oeabl
OK - Stopped application at context path /oeabl
```

See also

Start a Web application (enable) on page 213
The tcman command on page 200

Display global server resources (resources)

Purpose

List the global resources used by the core server.

To use this action, the Tomcat manager (manager.war) must be deployed on the instance and the instance must be running. You can deploy manager.war from $CATALINA_HOME/extras.

Syntax

```
tcman.sh resources [general_options]  -u user_name:password
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run tcman.sh help resources to see which general options are appropriate.

-u *user_name*:*password*

Pass a valid user name and a password for HTTP Basic access authentication.
(The default is -u tomcat:tomcat.)

Example

Display global resources for the running instance, acme1:

```
$: /psc/acme1/bin/tcman.sh resources -u tomcat:tomcat
OK - Listed global resources of all types
ServiceRegistry/ServiceRegistryFactory:com.progress.appserv.services.naming.ServiceRegistry
UserDatabase:org.apache.catalina.users.MemoryUserDatabase
```

Display Web application HTTP sessions (sessions)

Purpose

Display how many sessions are active for the specified Web application, categorized by their duration.

To use this action, the Tomcat manager (`manager.war`) must be deployed on the instance and the instance must be running. You can deploy `manager.war` from `$CATALINA_HOME/extras`.

Syntax

```
tcman.sh sessions [general_options] -u user_id:password app_name
```

Parameters

general_options

Specify one or more of the options that can be used with any TCMAN action.

`-u` *user_id*`:`*password*

Specify a valid user name and password for HTTP Basic access authentication. (The default is `-u tomcat:tomcat`.)

app_name

Specify the name of the web application to analyze for session information.

Example

Show the active sessions for the `manager` application deployed on the `acme1` instance:

```
/psc/acme1/bin/tcman.sh sessions -u tomcat:tomcat manager
OK - Session information for application at context path /manager
Default maximum session inactive interval 30 minutes
<1 minutes: 1 sessions
8 - <9 minutes: 2 sessions
9 - <10 minutes: 1 sessions
```

See also

List deployed applications (list) on page 205

Server actions

This section details the actions available for creating and monitoring server instances.

See also

The tcman command on page 200

Create an instance (create)

Purpose

Create a new instance of the core PAS server by running this action from `/bin` directory of the core server (`$CATALINA_HOME/bin/tcman.sh create`).

Syntax

```
tcman.sh create [general_options] [-f] [-p port_num] [-P port_num]
   [-s port_num] [-j port_num] [-m uid:pwd] [-W pathname] [-N instance_name]
   [-U user_id ] [-G group_id] [-Z {prod | dev}] base_path
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help create` to see which general options are appropriate.

`-f`

Copy all deployed web application archives (`.war` files) from `$CATALINA_HOME` to the new instance.

`-p` *port_num*

Specify the TCP port that listens for HTTP messages. The default is 8080.

`-P` *port_num*

Specify the TCP port that listens for HTTPS messages. The default is 8443.

`-s` *port_num*

Specify the TCP port to use to stop an instance. On Windows systems, you must specify a shutdown port. On UNIX, shutdown ports are optional.

`-j` *port_num*

Specify the TCP port that listens for AJP13 messages (an Apache protocol for handling requests from a web server to an application server). The default is 8009.

`-m` *`uid:pwd`*

Specify a user name and password that will be required to access Tomcat container-level security, which includes the `manager` and `oemanager` web applications. Replaces the defaults (`tomcat:tomcat`) in `/conf/tomcat-users.xml`.

`-W` *`pathname`*

Specify the directory where web applications will be deployed. The default is `$CATALINA_BASE/webapps`.

`-N` *`instance_alias`*

Specify an alias for the instance. If you do not specify an alias, the instance name will be the name of the directory where the instance is created.

Note:

All instances are automatically registered for tracking when they are created. However, for tracking to function, the instance name must not contain spaces or any of the following characters: `"[ . # | ] $ ? + = { / , }"`

`-U` *`user_id`*

Specify the user-id of the owner of all the files and directories of the instance. The default is the user-id of the current process. `-G` must be specified if you use this option.

`-G` *`group_id`*

Specify the group-id of the owner of all the files and directories of the instance. The default is the group-id of the current process. `-U` must be specified if you use this option.

`-Z` **{** `dev` **|** `prod` **}**

Specify the security model of the instance to development (`dev`) or secure (`prod`).

A typical use of this option is for testing web applications in a secure server environment before packaging and deploying.

Note: The `-Z prod` option does not create a production server. To actually create a production server, you must have a production server license.

`base_path`

Specify the pathname where you will create the instance.

Example

Create an instance of `/psc/pashome` in `/psc/acme1`:

```
$: /psc/pashome/bin/tcman.sh create -p 8501 -P 8601 -s 8701 /psc/acme1
Server instance acme1 created at /psc/acme1
```

See also

Delete an instance (delete) on page 219
Display and manage an instance's configuration (config) on page 220
Register an instance for tracking (register) on page 227
Start an instance (start) on page 231
Stop an instance (stop) on page 233

Delete an instance (delete)

Purpose

Remove the directory tree and all of the files in an instance. Alias tracking is disabled for servers that are removed.

To execute this action, the instance cannot be running.

Note: You cannot recover any files or directories removed by the `delete` action. Backup anything you want to save before launching this action.

Also note that you cannot use delete to remove the installed, root server (`$CATALINA_HOME`).

Syntax

```
tcman.sh delete [general_options] [-y] [base_path | alias_name]
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help delete` to see which general options are appropriate.

`-y`

Delete everything without prompting for confirmation.

base_path

Specify the pathname of the instance that you intend to delete.

alias_name

Refer to the instance that you intend to delete by its alias rather than its pathname.

Example

Delete the instance of `pashome` that was created in `/psc/acme3`:

```
$: /psc/pashome/bin/tcman.sh delete /psc/acme3
The following directory tree will be removed permanently:
 ( WARNING all deployed web applications will be DELETED!! )
/PAS/wrkdir/acme3
/PAS/wrkdir/acme3/conf
/PAS/wrkdir/acme3/temp
/PAS/wrkdir/acme3/common
/PAS/wrkdir/acme3/common/lib
/PAS/wrkdir/acme3/logs
/PAS/wrkdir/acme3/webapps
/PAS/wrkdir/acme3/webapps/ROOT
/PAS/wrkdir/acme3/webapps/ROOT/static
/PAS/wrkdir/acme3/webapps/ROOT/static/error
/PAS/wrkdir/acme3/webapps/ROOT/static/auth
/PAS/wrkdir/acme3/webapps/ROOT/META-INF
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/adapters
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/adapters/rest/PingService
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/adapters/soap
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/classes
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/classes/com
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/classes/com/progress
/PAS/wrkdir/acme3/webapps/ROOT/WEB-INF/classes/com/progress/appserv
/PAS/wrkdir/acme3/work
/PAS/wrkdir/acme3/bin
Type 'yes' to continue
yes
Delete operation complete
server  removed at /PAS/wrkdir/acme3
```

See also

Create an instance (create) on page 217
Display server instances (instances) on page 224
The tcman command on page 200

Display and manage an instance's configuration (config)

Purpose

View, add, update, or delete the property values specified in `../conf/appserver.properties` and in `../conf/catalina.properties`.

When you run `tcman.sh config` with no parameters, it displays the core Tomcat server's configuration, and all the properties in both `.../conf/appserver.properties` and `.../conf/jvm.properties`. Note, however, that you can only view `jvm.properties`. You cannot modify its contents with the `config` action.

Syntax

```
tcman.sh config [general_options]
[prop_name | prop_name=value | +prop_name=value | ~prop_name]
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help config` to see which general options are appropriate.

prop_name

Display the specified property and its value.

prop_name=value

Set the value of a property that exists in `.../conf/appserver.properties`.

+prop_name=value

Add a new property to `.../conf/appserver.properties` and set its value.

~prop_name

Remove the specified property from `.../conf/appserver.properties`.

Examples

Show the configuration and properties of `acme1`, an instance of the core server, `pashome`:

```
$: /psc/acme1/bin/tcman.sh config
Using CATALINA_BASE:   /psc/acme1
Using CATALINA_HOME:   /psc/pashome
Using CATALINA_TMPDIR: /psc/acme1/temp
Using JRE_HOME:        /tools/linuxx86_64/java64/jdk1.7.0_02/
Using CLASSPATH:       /psc/pashome/bin/bootstrap.jar:/psc/pashome/bin/tomcat-juli.jar
Using CATALINA_PID:    /psc/acme1/temp/catalina.pid
Server version: Apache Tomcat/7.0.42
Server built:   Jul 2 2013 08:57:41
Server number:  7.0.42.0
OS Name:        Linux
OS Version:     2.6.18-164.el5
Architecture:   amd64
JVM Version:    1.7.0_02-b13
...
```

Display the value of a single property:

```
$: /psc/acme1/bin/tcman.sh config psc.as.http.port
psc.as.http.port=8501
```

Update the value of a property that exists in the `appserver.properties` file and then check the value:

```
$: /psc/acme1/bin/tcman.sh config psc.as.http.port=6543
$: tcman.sh config psc.as.http.port
psc.as.http.port=6543
```

Add a new property/value pair to the `appserver.properties` file and check the value:

```
$: /psc/acme1/bin/tcman.sh config +my.home.dir=/home/jarhead
$: tcman.sh config my.home.dir
my.home.dir=/home/jarhead
```

Update the server certificate in the `catalina.properties` file (see https://docs.oracle.com/cd/E19879-01/821-0185/ablqz/index.html for information about generating, exporting, and downloading a new server certificate):

```
$: /psc/acme1/bin/tcman.sh config psc.as.https.keyalias=myNewCert
```

Remove a property/value pair from the `appserver.properties` file and check if deletion was successful:

```
$: /psc/acme1/bin/tcman.sh config ~my.home.dir
$: tcman.sh config my.home.dir
Property does not exist - my.home.dir
```

Caution: There are no restrictions to property removal. The server will be unable to start if you remove a property required by `conf/server.xml`.

Notes

- All property names are case sensitive.
- You cannot enter multiple property names (*`prop_name`*) on the command line to view, update, or add properties to the `appserver.properties` file.
- You cannot use the config action to update existing values or add new values to the `jvm.properties` file

Display or modify the server features of an instance (feature)

Purpose

View, enable, or disable the server features contained in the `/conf/server.xml` file of an instance.

When you run `tcman.sh feature` with no parameters, it displays a list of the features (and their current status) that you can enable or disable. You can also display the status of a single server feature. After viewing the status of a feature, you can use `tcman.sh feature` to change its setting.

Syntax

```
tcman.sh feature [general_options] [feature_name [={on | off}]]
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help feature` to see which general options are appropriate.

feature_name

Specify one of the features defined in an instance's `conf/server.xml` file. Running `tcman.sh feature` without *feature_name* displays a list of all the features.

`on`

Enables the named feature.

`off`

Disables the named feature.

Example

Display the list of server feature settings for `acme1`, enable AJP13 (Apache JServ Protocol. version 1.3), and verify that the feature is enabled:

```
$: /psc/acme1/bin/tcman.sh feature
SecurityListener=off
JMXLifecycle=off
PSCRegistry=on
HTTP=onHTTPS=on
AJP13=off
Cluster=off
UserDatabase=on
JAASRealm=off
LDAPRealm=off
PASInstrument=off
RemoteHostValve=on
RemoteAddrValve=onSingleSignOn=on
AccessLog=on
CrawlerSessionManager=on
StuckSessionValve=on

$: /psc/acme1/bin/tcman.sh feature AJP13=on


$: /psc/acme1/bin/tcman.sh feature AJP13
AJP13=on
```

Notes

- Server features for instances are set in `$CATALINA_BASE/conf/server.xml`. You can change feature status by manually editing this file. However, it is safer to use `tcman.sh feature` to avoid corrupting the file with erroneous entries.
- Run `tcman.sh feature` when the instance is offline.

See also

Display and manage an instance's configuration (config) on page 220
Stop an instance (stop) on page 233

The tcman command on page 200

Clean up or archive server log files (clean)

Purpose

Truncate, move, or delete the log files located in the `/logs` directory of the core server or instance. If the server is running, clean truncates log files to zero length. If the server is not running, clean deletes the log files from the file system.

You have the option to save log files to a subdirectory of `/logs`.

Syntax

```
tcman.sh clean [general_options] [-A]
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help clean` to see which general options are appropriate.

`-A`

Archive log files to a subdirectory of `$CATALINA_BASE/logs`. The directory is automatically named with a month-day-year-second (*MM-DD-YYYY-ss*) time-stamp format. If the server is not running, the files in `$CATALINA_BASE/logs` are deleted.

Example

Archive the log files of `acme1`, an instance of the core server `pashome`, and save to a file:

```
/psc/pashome/tcman.sh clean -I acme1 -A
```

See also

The tcman command on page 200

Display server instances (instances)

Purpose

Show the names and locations of the instances created from the PAS installed in `$CATALINA_HOME` by displaying the contents of the file where instances are registered for tracking.

By default, instances are registered for tracking `$CATALINA_HOME/conf/instances.{windows|.unix}`. The file name extension indicates the OS platform where the PAS server is installed.

Syntax

```
tcman.sh instances [general_options]
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help instances` to see which general options are appropriate.

Output format

The following is the format of the output from a TCMAN `instances` action:

```
alias-name | full-file-path | type | state
```

alias-name

The user-defined name for the instance.

full-file-path

The location, in the OS file system, of the instance's root directory.

type

The designation of the server instance type (for example: `instance`, `service`, . . .).

state

An indication of the instance's validity. `OK` is returned for a valid server and `invalid` is returned for a corrupted or non-existant server.

Example

Display the instances of the core server installed in `/psc/pashome`:

```
/psc/pashome/bin/tcman.sh instances
acme1 | /psc/wrk/acme1 | instance | ok
acme2 | /psc/wrk/acme2 | instance | ok
```

Notes

- By default, instances are registered when you execute a `$CATALINA_HOME/bin/tcman{.sh|.bat} create` action, which automatically adds instance entries to an `instances` file. TCMAN removes instance entries from the file when you execute a `delete` action.

  You can manually add or remove instance entries from `instances` by using the `register` or `unregister` actions.

- By default, the name and location of the file where instances are registered is `$CATALINA_HOME/conf/instances.{windows|.unix}`.

  You can change the location of the instance registration file by adding and setting the `psc.as.instdir` property in the `appserver.properties` file. Use the TCMAN config action as in the following example:

  ```
  tcman.sh config '+psc.as.instdir=PATH'
  ```

  where *PATH* is a path name or an environment variable.

  You can also change the location and/or name of instance registration files by setting the environment variables, `PAS_AS_INSTANCE_DIR`, and `PAS_AS_INSTANCE_FILE`.

See also

Register an instance for tracking (register) on page 227
Stop tracking an instance (unregister) on page 236
The tcman command on page 200

List process ids (plist)

Purpose

List the process ids for all the processes that are running under an instance.

Syntax

```
tcman.sh plist [general_options] [-f]
```

Parameters

general_options

Specify one or more of the options that can be used with any TCMAN action. Run `tcman.sh help plist` to see which general options are appropriate.

`-f`

Display verbose output. The output is indented and uses the plus (+) character to indicate parent-child relationships.

Examples

Display process id's for the running instance, `acme1` using the `-v` and `-f` options:

```
/psc/acme1/bin/tcman.sh plist -v
info: showing process ids for server 5942
5942 5963 5975 5988 6001 6015

/psc/acme1/bin/tcman.sh plist -f
5942
 +5963
 +5975
 +5988
 +6001
 +6015
```

Notes

The plist action is useful for administrative tasks such as:

- Checking if an instance is stopped or running. If it is running, it returns the process id of the instance. If it is stopped, 0 is returned.
- Using the output (which is easily parsable) in administrative scripts.
- If the instance's applications have started sub-processes, multiple process ids will be returned.
- If multiple process ids are listed, the instance is always listed first.

See also

Show Windows process information (showproc) on page 230

Register an instance for tracking (register)

Purpose

Register an instance for tracking purposes.

Note:

Instances are automatically registered for tracking when you execute a `create` action. You use the `register` action to restart tracking on instances after tracking was stopped.

A typical use for unregistering and then re-registering an instance is to make configuration changes when moving instances from one location (core server) to another. The `register` action enables tracking and also updates the value of `CATALINA_HOME` in all of the executable scripts in the instance's `/bin` directory to refer to the new core server.

Syntax

```
tcman.sh register alias_name instance_path
```

Parameters

alias_name

Specify a meaningful name for the instance. The alias name must be unique in the instances file.

instance_path

Specify the OS file system path to where the instance exists. This value will be expanded into a fully qualified OS directory path and will be verified to exist.

Example

Track `test1`, which is an alias for the instance `/psc/acme1`:

```
/psc/pashome/bin/tcman.sh register test1 /psc/acme1
```

Notes

When you register an instance for tracking or create a new instance with the `create` command, an entry is created in the core Progress Application Server's `$CATALINA_HOME/conf/instances.[unix|windows]` file.

The `instances.[unix|windows]` file is a simple text file, which can be manually edited (with care) in the event that it becomes out of date. The format for entries is:

```
instance_name = base_path
```

An `instances.unix` file uses Unix OS file path syntax (forward slashes), and an `instances.windows` file uses Windows OS file path syntax (backslashes) to specify *base_path*.

Also note that in an instances file:

- Any line starting with a pound-sign (#) is a comment line.
- Blank lines are skipped.
- Instance names cannot contain spaces or any of the following characters: "`[ . # | ] $ ? + = { / , }`"

See also

Stop tracking an instance (unregister) on page 236
The tcman command on page 200

Register and manage an instance as a Windows service (service)

Purpose

(Windows only) Registers or unregisters an instance as a Windows service. After an instance is registered, you can start, stop, or check the status of the service with this action.

Note: Before you register an instance as a Windows service, you must install JDK 1.8 (for example, jdk1.8.0_66) and set the environment variable `JAVA_HOME=C:\Program Files\Java\jdk1.8.0_66`.

Syntax

```
tcman.bat service [general_options] alias_name { register | unregister | start
| stop | status }
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.bat help service` to see which general options are appropriate.

alias_name

A required parameter that specifies the name of a PAS instance that was created using `tcman create`.

`register`

Create a new Windows service that runs the named PAS instance *alias_name*..

Set the `PR_DISPLAYNAME` and/or `PR_DESCRIPTION` variables to change the display name and description of the PAS instance service that appears in the Windows Service utility (**Services** tab of the Task Manager). The defaults for these variables are:

Variable	Default
`PR_DISPLAYNAME`	Progress Application Server *alias_name*
`PR_DESCRIPTION`	Progress Application Server (Tomcat 7) – http://www.progress.com

Set these variables before you register the instance. For example, if you wanted to change the defaults for `oepas1`:

```
set PR_DISPLAYNAME=PAS ROOT Server
set PR_DESCRIPTION=Progress Application Server
tcman service oepas1 register
```

`unregister`

Delete the Windows service that runs the named PAS instance *alias_name*

`start`

Start the Windows service that corresponds to the named PAS instance *alias_name*. The Windows service may also be started using the Windows service console or the SC command line utility.

`stop`

Stop the Windows service that corresponds to the named PAS instance *`alias_name`*. The Windows service may also be stopped using the Windows service console or the SC command line utility.

`status`

The registration status of the Windows service corresponding to the named PAS instance *`alias_name`*. The Windows service's status may be monitored using the Windows service console or SC command line utility

Example

Register the default instance oepas1 as a Windows, then start, check status, stop, and unregister:

```
tcman service oepas1 register
  oepas1 service is registered

tcman service oepas1 start
  oepas1 started

tcman service oepas1 status
  Service oepas1 is running

tcman service oepas1 stop
  oepas1 is stopped

tcman service oepas1 unregister
  oepas1 is unregistered
```

Note

Be sure that the instance is not running before you attempt to register/unregister it.

Show Windows process information (showproc)

Purpose

(Windows only) Show information about the process specified by a process id.

Syntax

```
tcman.bat showproc [general_options] [process-id]
```

Parameters

`general_options`

Specify one or more of the options that can be used with any TCMAN action. Run `tcman.sh help showproc` to see which general options are appropriate.

process-id

The numerical identifier of a Windows process. You can obtain a list of process ids by running the TCMAN `plist` action.

Examples

Display process id's for the running instance, `acme1`, then specify process ids to show detailed information.

```
/psc/acme1/bin/tcman.bat plist -v
info: showing process ids for server with window title 13332
13332 14240

/psc/acme1/bin/tcman.bat showproc 13332
ProcesName                : java
SessionId                 : 2
StartTime                 : 10/04/2015 16:29:42
Threads                   : 26
TotalProcessorTime        : 00:00:19.9213277
UserProcessTime           :
CPU (seconds)             : 19.9213277
Description               : Java(TM) Platform SE binary
Path                      : C:\Progress\OpenEdge\jdk\bin\java.exe

/psc/acme1/bin/tcman.bat showproc 14240
ProcesName                : _mproapsv
SessionId                 : 2
StartTime                 : 10/04/2015 16:29:54
Threads                   : 7
TotalProcessorTime        : 00:00:00.3744024
UserProcessTime           :
CPU (seconds)             : 0.3744024
Description               : OpenEdge AppServer (Multi-thread)
Path                      : C:\Progress\OpenEdge\bin\_mproapsv.exe
```

See also

List process ids (plist) on page 226

Start an instance (start)

Purpose

Start an instance of a PAS, optionally in debug mode.

Syntax

```
tcman.sh start [general_options] [-D|-J]
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help start` to see which general options are appropriate.

`-D`

Start the server in Tomcat debug mode. `-D` overrides the `-J` option.

`-J`

Start the server in debug mode using the JDPA (Java Platform Debugger Architecture) APIs for debugging. `-J` cannot be used if the `-D` option is specified.

Before you run a server with the `-J` option, you must define a port for the JDPA debugger by setting the `JDPA_ADDRESS` environment variable to a unique TCP network port number.

Example

Start the server in `/psc/acme1`, which is an instance of the core server in `/psc/pashome`:

```
/psc/acme1/bin/tcman.sh start
Using CATALINA_BASE:   /psc/acme1
Using CATALINA_HOME:   /psc/pashome
Using CATALINA_TMPDIR: /psc/acme1/temp
Using JRE_HOME:        /tools/linuxx86_64/java64/jdk1.7.0_02/
Using CLASSPATH:       /psc/pashome/bin/bootstrap.jar:/psc/pashome/bin/tomcat-juli.jar
Using CATALINA_PID:    /psc/acme1/temp/catalina.pid
```

Notes

- When the TCMAN utility starts the server, it verifies the creation of the OS process and then records the server's process-id in a `.pid` file. The location of the `.pid` file is:

OS	PID File Path
UNIX	`$CATALINA_BASE/temp/catalina-`*`instance_name`*`.pid`
Windows	`$CATALINA_BASE\logs\catalina-`*`instance_name`*`.pid`

- You can obtain the process id of a server by running the TCMAN `env` action.

See also

Stop an instance (stop) on page 233
The tcman command on page 200

Stop an instance (stop)

Purpose

Stop a running instance, either gracefully or forcibly.

Note: TCMAN supports stopping a server instance that is not configured with a shutdown port.

On UNIX platforms stopping the running server instance is accomplished by sending a UNIX signal to the PAS process. Therefore, the administrator's process must have the UNIX permissions to signal the PAS process. On Windows platforms, the instance is identified using an OS process id that is used to stop server processes.

Syntax

```
tcman.sh stop [general_options] [-F[-w seconds]]
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help stop` to see which general options are appropriate.

`-F`

Kill the sever process if it does not stop after a default wait time (5 seconds on UNIX, 10 seconds on Windows). Change the default wait interval by using the `-w` option.

`-w` *seconds*

Optionally specify the number of seconds to wait before killing a server process.

Example

Stop the server in `/psc/acme1`, which is an instance of the core server in `/psc/pashome`:

```
/psc/acme1/bin/tcman.sh stop
Using CATALINA_BASE:   /psc/acme1
Using CATALINA_HOME:   /psc/pashome
Using CATALINA_TMPDIR: /psc/acme1/temp
Using JRE_HOME:        /tools/linuxx86_64/java64/jdk1.7.0_02/
Using CLASSPATH:       /psc/pashome/bin/bootstrap.jar:/psc/pashome/bin/tomcat-juli.jar
Using CATALINA_PID:    /psc/acme1/temp/catalina.pid
```

Notes

- TCMAN supports stopping a server instance that is not configured with a shutdown port.

On UNIX platforms stopping the running server instance is accomplished by sending a UNIX signal to the PAS process. Therefore, the administrator's process must have the UNIX permissions to signal the PAS process. On Windows platforms, the instance is identified using an OS process id that is used to stop server processes.

The following is an example a message you would see after a forced shut down with no shut down port:

```
Sep 23, 2013 4:10:47 PM org.apache.catalina.startup.Catalina stopServer
SEVERE: No shutdown port configured. Shut down server through OS signal.
Server not shut down.
Killing Tomcat with the PID: 14230
```

- Process ids are stored in the following locations:

OS	PID File Path
UNIX	`$CATALINA_BASE/temp/catalina-`*`instance_name`*`.pid`
Windows	`$CATALINA_BASE\logs\catalina-`*`instance_name`*`.pid`

- You can also obtain the process id of a server by running the TCMAN `env` action.

See also

Start an instance (start) on page 231
The tcman command on page 200

Display server, OS, and runtime version information (version)

Purpose

Show the Apache Tomcat runtime version and OS information for an instance.

To execute this action, the instance cannot be running

Syntax

```
tcman.sh version [general_options]
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help version` to see which general options are appropriate.

Example

Display the server and runtime information for `acme1`, an instance of the core server installed in `/psc/pashome`:

```
$: /psc/pashome/bin/tcman.sh version -I acme1
Using CATALINA_BASE:   /psc/acme1
Using CATALINA_HOME:   /psc/pashome
Using CATALINA_TMPDIR: /psc/acme1/temp
Using JRE_HOME:        /tools/linuxx86_64/java64/jdk1.7.0_02/
Using CLASSPATH:
/psc/pashome/bin/bootstrap.jar:/users/doc/agarbacz/psc/pashome/bin/tomcat-juli.jar
Using CATALINA_PID:    /psc/acme1/temp/catalina.pid
Server version: Apache Tomcat/7.0.42
Server built:   Jul 2 2013 08:57:41
Server number:  7.0.42.0
OS Name:        Linux
OS Version:     2.6.18-164.el5
Architecture:   amd64
JVM Version:    1.7.0_02-b13
JVM Vendor:     Oracle Corporation
```

See also

Display OS and server information (info) on page 206
The tcman command on page 200

Test a server configuration (test)

Purpose

Displays information on the configuration and environment of an instance. It also displays information about error conditions.

The `test` action starts a server (instance), loads all the configuration files, and then displays information. The instance is stopped, exiting gracefully even if there is an error condition.

To execute this action, the instance cannot be running.

Syntax

```
tcman.sh test [general_options]
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help test` to see which general options are appropriate.

Example

Run a test of the configuration of `acme1`, which is an instance of the core server installed at `/psc/pashome`:

```
$: /psc/pashome/bin/tcman.sh -I acme1 test
Using CATALINA_BASE:   /psc/acme1
Using CATALINA_HOME:   /psc/pashome
Using CATALINA_TMPDIR: /psc/acme1/temp
Using JRE_HOME:        /tools/linuxx86_64/java64/jdk1.7.0_02/
Using CLASSPATH:       /psc/pashome/bin/bootstrap.jar:/psc/pashome/bin/tomcat-juli.jar
Using CATALINA_PID:    /psc/acme1/temp/catalina.pid
. . .
```

Notes

The `test` action is particularly useful for testing to verify that a server will start and run properly after you make changes to configuration and properties files.

See also

The tcman command on page 200

Stop tracking an instance (unregister)

Purpose

Stop tracking an instance by removing the instance's entry from the `$CATALINA_HOME/conf/instances.[unix|windows]` file.

Note:

You use the `register` action to restart tracking on instances after tracking was stopped with `unregister` .

A typical use for unregistering and then re-registering an instance, is to make configuration changes when moving instances from one location, or core server, to another. The `register` action not only enables tracking, it also updates the value of `CATALINA_HOME` in all of the executable scripts in the instance's `/bin` directory to refer to the new core server.

Syntax

```
tcman.sh unregister alias_name
```

Parameters

alias_name

Specify the alias name of the instance that you want to stop tracking. The alias name must exist in an `instances.[unix|windows]` file.

Example

Stop tracking `test1`, which is an instance of `/psc/pashome`:

```
/psc/pashome/bin/tcman.sh unregister test1
```

See also

Register an instance for tracking (register) on page 227
The tcman command on page 200

Create a Tomcat worker configuration file (workers)

Purpose

Create a preliminary `worker.properties` file that supports the configuration of supporting servers (workers) for a Progress Application Server (PAS) instance.

in the Apache Reference Guide, a worker is defined as an "instance that is waiting to execute servlets or any other content on behalf of some web server." In the context of the Progress Application Server, a worker is a server that is called by a PAS instance to perform a specific task. Typically,you would define worker instances to manage proxies, load balancing, clusters, or status monitoring. (For links to information on this functionality, see the Apache Tomcat Documentation Index.) There are probably other situations where you could improve the performance of a server instance by configuring worker instances to handle specific processing tasks.

In Apache Tomcat, workers are configured in a `worker.properties` file. The protocol implemented for communication between servers and workers is the Apache JServ Protocol (version 1.3, referred to as AJP13).

In TCMAN, the `workers` action adds the definitions of registered PAS instances to the content of the `$CATALINA_HOME/extras/workers.template` file and puts the result in `$CATALINA_HOME/temp/worker.properties`. The template file supplies a set of common directives that are referenced by all of the defined PAS instances. Individual instance definitions contain only the properties that are unique to the instance, such as the AJP13 network connection port. (See Table 18: worker.properties example on page 239.)

The `/temp/worker.properties` created by the `workers` action is a preliminary configuration file that you will probably need to modify to implement your deployment. See The Apache Tomcat Connector-Reference Guide for more information about configuring workers.

Syntax

```
$CATALINA_HOME/tcman.sh workers [general_options] [worker_list]
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help worker` to see which general options are appropriate.

`worker_list`

A comma separated list of instance names and/or keywords. The keywords are:

status	Include an instance that has been implemented as a status server
lb	Include an instance that has been implemented as a load balance server
home	Include the CATALINA_HOME core server
all	Include all registered instances

If no *`worker_list`* is specified, the default worker list (all instances registered to CATALINA_HOME) will be added. If no instances have been created, then the default *`worker_list`* is CATALINA_HOME.

Examples

Assume there are:

- Two registered instances (`piw1` and `piw2`) that serve Web applications
- A Tomcat load balancer instance (`jklb`) that distributes the workload between `piw1` and `piw2`
- A status instance (`jkstatus`) that is used to monitor the runtime status of `piw1` and `piw2`

The following are examples of worker-lists showing various combinations of keywords and instances, and the resulting content in `$CATALINA_HOME/temp/worker.properties`:

Table 17: worker-list keywords

worker-list	Resulting content in worker.properties
`blank`	Default entries from `worker.template` plus entries for `piw1` and `piw2`
`piw1`	Default entries from `worker.template` plus an entry for `piw1`
`all`	Default entries from `worker.template` plus entries for `piw1` and `piw2`
`home`	Default entries from `worker.template` plus an entry for core server (CATALINA_HOME)
`home, all`	Default entries from `worker.template` plus an entry for core server, (CATALINA_HOME), and entries for for `piw1` and `piw2`
`lb, status`	Default entries from `worker.template` plus entries for `jklb`, `jstatus`, `piw1`, and `piw2` **Note:** When no registered instance is specified, all registered instances are automatically included.
`lb, status, home, all`	Default entries from `worker.template` plus entries for `jklb`, `jstatus`, the core server (CATALINA_HOME),`piw1`, and `piw2`

The following is an example `workers.properties` file that includes entries for instances `piw1` and `piw2`:

Table 18: worker.properties example

```
# worker properties for instances rooted at U:\vobs\dlc\servers\pasw
# build date: 03/24/2014 11:09:23

# List of worker server instances
worker.list=piw1,piw2

#
# Global properties
#
# worker.maintain=60

#
# Common worker properties referenced by individual workers
#
worker.common.type=ajp13
worker.common.host=${psc.as.host.name}
worker.common.socket_timeout=10
worker.common.connect_timeout=10000
worker.common.socket_keepalive=true
worker.common.ping_mode=I
worker.common.ping_timeout=10000
worker.common.connect_timeout=0
worker.common.retry_interval=100
worker.common.recovery_options=7
# worker.common.connection_ping_interval=10000
# worker.common.fail_on_status=0# worker.common.max_packet_size=8192
# worker.common.recover_time=60

# properties for alias piw1 with jvmRoute piw1
worker.piw1.port=9996
worker.piw1.reference=worker.common

# properties for alias piw2 with jvmRoute piw2
worker.piw2.port=9996
worker.piw2.reference=worker.common
```

Notes

- The `tcman workers` action must be run from the PAS installation's `$CATALINA_HOME/bin directory`.
- The `/extras/workers.template` file can be modified to adjust existing properties or to add additional static information. However, you cannot replace the common properties with a unique set of properties for each defined server.

General actions

This section details the actions available for displaying help and server runtime environment information.

See also

The tcman command on page 200

Display help (help)

Purpose

Display summary or detailed help for all TCMAN actions, property names, and server features.

Syntax

```
tcman.sh help [action | property | feature]
```

Parameters

action

Show the syntax and options of the specified action. If no action is specified, show a list of all actions and the general options.

property

Show the settings for specified property.

feature

Show if the specified feature is enabled or disabled.

Example

Display the usage help for the `create` action:

```
$: tcman.sh help create
usage: tcman create [options] -p <http-port> [instance-opts] <new-base-path>
    instance-opts:
       [-s <shutdown-port>]
       [-P <https-port>]
       [-j <ajp13-port>]
       [-W <web-apps-dir>]
       [-N <inst-alias-name>]
       [-U <file-owner> -G <file-group>]

   general options:
      -u uid:pwd  pass uid and pwd for HTTP BASIC authentication
      -v          print verbose output
      -M url      override the CATALINA_BASE manager's URL with
                  <{http|https}://<host>:<port>/<mgr-app>
      -B          override CATALINA_BASE environment setting
      -n          debug run action but do not execute changes
```

Display runtime environment information (env)

Purpose

Show details about a server's state.

Syntax

```
tcman.sh env [general_options] [keyword]
```

Parameters

general_options

Specify one or more of the general TCMAN options. Run `tcman.sh help env` to see which general options are appropriate.

keyword

Specify one or more keywords that represent the name of the state that you want to view. If no keyword is specified, then all of the state information is displayed.

Keywords include:

Keyword	Description
`running`	Indicate if a server is running (`1`) or not running (`0`).
`mgrurl`	Display the URL of the manager application.
`type`	Display the server type.
`alias`	Display the server's alias.
`parent`	Display the pathname of the parent of an instance.
`tracking`	Indicate if tracking is on (`1`) or off (`0`).
`http`	Display the server's http port number.
`https`	Display the server's https port number.
`shut`	Display the server's shutdown port number. A value of `-1` indicates that there is no shutdown port.
`pid`	Display the server's process id. A hyphen (`-`) indicates that the server is not running.

Example

Display all of the state information for the instance created in `/psc/acme1`:

```
/psc/acme1/bin/tscman.sh env
catalina home:     /psc/pashome
catalina base:     /psc/acme1
java home:         /tools/linuxx86_64/java64/jdk1.7.0_02/
jre home:
manager http port: 8501
manager https port:8601
manager shut port: 8701
manager URL:       http://localhost:8501/manager
config type:       instance
config alias:      acme1
config parent:     /psc/pashome
server running:    0
instance tracking: 1
instance file:     /psc/pashome/conf/instances.unix
server process-id: -
```

11

REST API Reference for oemanager.war

The `oemanager.war` Java Web application provides a REST API for remote management and monitoring of ABL applications deployed on a Progress Application Server for OpenEdge instance. It is similar to the administration API supported by the Tomcat Web server's JMX interface, but it uses JSON input/output payloads.

Requirements

To enable REST API access, either `oemanager.war` or `tcmanager.war` must be deployed.

Examples

The examples in this section show the use of `curl`, which is a common open software utility used in command lines or scripts to transfer data. You can, however, employ the PAS for OpenEdge REST management APIs with many other tools and utilities. For more information about `curl`, see: http://curl.haxx.se/docs/.

For details, see the following topics:

- Session management
- Agent management
- Transport management
- Application management
- Monitoring memory use

Session management

This section describes the REST services that are available for managing Progress Application Server for OpenEdge sessions.

See also

Get session information on page 244
Get client information on page 245
Get current request information on page 247
Get configuration properties on page 248
Update configuration properties on page 249
Get runtime metrics on page 250
Reset runtime metrics on page 251
Cancel a request on page 252
Terminate a session on page 254

Get session information

Description

Query an ABL application to get a list of all its sessions including information on client and agent connections.

HTTP Operation

```
GET
```

URI

```
//host_name:port/oemanager/applications/ABL_app_name/sessions
```

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
404 Resource Not Found
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v http://localhost:16680/oemanager/applications/oepas1/sessions
```

Request body example

NA

Response body example


```
{
"operation":"GET CLIENT SESSIONS",
"result":
    {
    "OEABLSession":
    [
        {
         "elapsedTimeMs":3343,
         "bound":false,
         "requestState":
         "RUNNING","requestID":"",
         "agentConnInfo":
             {
             "state":"RESERVED",
             "localAddr":"\/127.0.0.1:60464",
             "agentID":"AG-\/5vs3eApTMiPV5M+W47T3w",
             "connID":"AC-nRJgbuKtSNWcy6YWQknTbg",
             "connPoolID":"CP-30BInu3FRwOGQB0u0LwJ\/g",
             "agentAddr":"localhost\/127.0.0.1:62004"},
             "ablSessionID":"5",
             "sessionState":"RESERVED",
             "agentID":"AG-\/5vs3eApTMiPV5M+W47T3w",
             "clientConnInfo":
                 {
                 "elapsedTimeMs":5214,
                 "requestProcedure":"ngasPing.p",
                 "requestUrl":"http:\/\/localhost:16680\/apsv",
                 "clientName":"172.21.75.148",
                 "httpSessionId":"732277EE12485DDEC3A937DD30D7EF3267B3FE669C76.pas1",
                 "executerThreadId":"catalina-exec-8",
                 "reqStartTimeStr":"2014-09-17T08:41:48.138-0400",
                 "requestID":"ngasPing.p",
                 "adapterType":"HTTP",
                 "sessionID":"Iphz68lUQKexuHibyy6S+A"
                 },
             "adapterType":"HTTP",
             "sessionPoolID":"SP-RFplWI9\/Rta9j1pp+WcRyQ",
             "sessionID":"Iphz68lUQKexuHibyy6S+A",
             "sessionType":"SESSION_MANAGED",
             "lastAccessStr":"2014-09-17T08:41:50.006-0400"
        }
    ]
    },
    "outcome":"SUCCESS",
    "errmsg":"",
    "versionStr":"PASOE 11.5.0",
    "versionNo":1
}
```


Get client information

Description

Query an ABL application to get a information on client connections.

HTTP Operation

```
GET
```

URI

```
//host_name:port/oemanager/applications/ABL_app_name/clients [?timeoutms=n]
```

`timeoutms`

The period of time, specified in milliseconds (ranging from 1 to *n*), to report on client connections, including hung clients.

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line examples

```
curl -X GET -v http://localhost:16680/oemanager/applications/oepas1/clients

curl -X GET -v
http://localhost:16680/oemanager/applications/oepas1/clients?timeoutms=100000
```

Request body example

NA

Response body example


```
{
"operation":"GET CLIENT CONNECTIONS",
"result":
    {
    "ClientConnection":
        [
        {
        "elapsedTimeMs":427133,
        "requestProcedure":"ngasPing.p",
        "requestUrl":
        "http:\/\/localhost.com:16680\/apsv",
        "clientName":"172.21.75.148",
        "httpSessionId":"732277EE12485DDEC3A937DD30D7EF3267B3FE669C76.pas1",
        "executerThreadId":"catalina-exec-8",
        "reqStartTimeStr":"2014-09-17T08:41:48.138-0400",
        "requestID":"ngasPing.p",
        "adapterType":"HTTP",
        "sessionID":"Iphz68lUQKexuHibyy6S+A"
        }
        ]
    },
    "outcome":"SUCCESS",
    "errmsg":"",
    "versionStr":"PASOE 11.5.0",
    "versionNo":1
}
```

Get current request information

Description

Query an ABL application to get a list of all the requests currently in process.

HTTP Operation

```
GET
```

URI

```
//host_name:port/oemanager/applications/ABL_app_name/requests
```

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v http://localhost:16680/oemanager/applications/oepas1/requests
```

Request body example

NA

Response body example

```
{
"operation":"GET ACTIVE REQUESTS ON SERVER",
"result":
    {
    "Request":
        [
        {
        "requestElapsedTime":1889,
        "requestStartTimeStamp":"2014-09-17T10:11:26.934-0400",
        "requestState":"RUNNING",
        "requestID":"xfJqPACGU4jqE4pd0Auzwg",
        "sessionID":"8DvBtOplRHijAj7NcSAQKw"
        }
        ]
     },
     "outcome":"SUCCESS",
     "errmsg":"",
     "versionStr":"PASOE 11.5.0",
     "versionNo":1
}
```

Get configuration properties

Description

Query the `conf/openedge.properties` file to get the configuration properties of an application. The information is specified in the `AppServer.SessMgr.`*`App_name`* section of `openedge.properties`.

HTTP Operation

`GET`

URI

`//`*`host_name`*`:`*`port`*`/oemanager/applications/`*`App_name`*`/properties`

Media type

`application/vnd.progress+json`

Response codes

```
200 Success
404 Resource Not Found
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v http://localhost:16680/oemanager/applications/oepas1/properties
```

Request body example

NA

Response body example


```
{
"operation":"GET SESSION-MGR PROPERTIES",
"result":
    {
    "agentLogThreshold":"0",
    "requestWaitTimeout":"15000",
    "agentLogEntryTypes":"ASPlumbing,DB.Connects",
    "idleAgentTimeout":"300000",
    "idleConnectionTimeout":"300000",
    "socketTimeout":"3000",
    "agentListenerTimeout":"300000",
    "applications":"oepas1",
    "numInitialAgents":"1",
    "agentLogFile":"\/wrk\/pas1\/logs\/oepas1.agent.log",
    "collectMetrics":"1",
    "agentExecFile":"\/localhost\/dlc\/bin\/_mproapsv",
    "idleResourceTimeout":"0",
    "allowRuntimeUpdates":"0",
    "connectionWaitTimeout":"3000",
    "maxAgents":"10",
    "publishDir":"\/wrk",
    "maxConnectionsPerAgent":"16",
    "idleSessionTimeout":"300000",
```

```
    "agentStartupParam":"-T \/wrk\/pas1\/temp",
    "statusEnabled":"1",
    "maxABLSessionsPerAgent":"200",
    "agentNumLogFiles":"3",
    "agentLoggingLevel":"3",
    "ipver":"IPv4"
    },
"outcome":"SUCCESS",
"errmsg":"",
"versionStr":"PASOE 11.5.0",
"versionNo":1
}
```

Update configuration properties

Description

Update an application's configuration properties in the `AppServer.SessMgr.`*`App_name`* section of the `conf/openedge.properties` file.

HTTP Operation

`PUT`

URI

`//host_name:port/oemanager/applications/`*`App_name`*`/properties`

Media type

`application/vnd.progress+json`

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X PUT -v http://localhost:16680/oemanager/applications/oepas1/properties
-d '{"agentLogThreshold":"1"}' -H "Content-Type: application/vnd.progress+json"
```

Request body example

```
{
"operation":"SET BROKER PROPERTIES",
"result":
    {
    "agentLogThreshold":"1",
    "requestWaitTimeout":"15000",
    "agentLogEntryTypes":"ASPlumbing,DB.Connects",
    "idleAgentTimeout":"300000",
    "idleConnectionTimeout":"300000",
    "socketTimeout":"3000",
    "agentListenerTimeout":"300000",
    "applications":"oepas1",
    "numInitialAgents":"1",
```

```
    "agentLogFile":"\/scratch\/prmundra\/17Sept\/wrk\/pas1\/logs\/oepas1.agent.log",
    "collectMetrics":"1",

"agentExecFile":"\/view\/rdl115_linuxx86_64\/vobs_prgs\/linuxx86_64\/dlc\/bin\/_mproapsv",

    "idleResourceTimeout":"0",
    "allowRuntimeUpdates":"0",
    "connectionWaitTimeout":"3000",
    "maxAgents":"10",
    "publishDir":"\/scratch\/prmundra\/17Sept\/wrk",
    "maxConnectionsPerAgent":"16",
    "idleSessionTimeout":"300000",
    "agentStartupParam":"-T \/scratch\/prmundra\/17Sept\/wrk\/pas1\/temp",
    "statusEnabled":"1",
    "maxABLSessionsPerAgent":"200",
    "agentNumLogFiles":"3",
    "agentLoggingLevel":"3",
    "ipver":"IPv4"
    },
"outcome":"SUCCESS",
"errmsg":"",
"versionStr":"PASOE 11.5.0",
"versionNo":1
}
```


Response body examples

On success:


```
{
 "operation":"SET SESSION-MGR PROPERTIES",
 "result":"NA",
 "outcome":"SUCCESS",
 "errmsg":"NA",
 "versionStr":"PASOE 11.5.0",
 "versionNo":1
}
```


On failure:


```
{
 "Operation":"SET SESSION-MGR PROPERTIES",
"Outcome":"FAILURE",
"ErrorMsg":"Invalid property name: "InvalidPropName". (PRPMsg009)"
}
```


Get runtime metrics

Description

Query the session manager for runtime metrics of an ABL application.

HTTP Operation

GET

URI

```
//host_name:port/oemanager/applications/ABL_app_name/metrics
```

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v http://localhost:16680/oemanager/applications/oepas1/metrics
```

Request body example

NA

Response body example

```
{
"operation":"GET SESSION-MGR METRICS",
"result":
    {
    "writeErrors":0,
    "concurrentConnectedClients":0,
    "maxQueueDepth":0,
    "maxConcurrentClients":0,
    "readErrors":0,
    "reserveConnectionTimeouts":0,
    "timesQueued":0,
    "Writes":0,
    "Requests":0,
    "Reads":0,
    "avgQueueDepth":0,
    "type":"OE_BROKER",
    "accessTime":"2014-09-17T10:26:12.693-04:00"
    },
"outcome":"SUCCESS",
"errmsg":"",
"versionStr":"PASOE 11.5.0",
"versionNo":1
}
```

Reset runtime metrics

Description

Reset runtime metrics of the session manager.

HTTP Operation

DELETE

URI

```
//host_name:port/oemanager/applications/ABL_app_name/metrics
```

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X DELETE -v http://localhost:16680/oemanager/applications/oepas1/metrics
```

Request body example

NA

Response body example

```
{
"operation":"RESET SESSION-MGR METRICS",
"result":"","outcome":
"SUCCESS","errmsg":"",
"versionStr":"PASOE 11.5.0",
"versionNo":1
}
```

Cancel a request

Description

Cancel the request identified by a Request ID and a Session ID. You can obtain Request IDs and Session IDs by executing the `oemanager/applications/`*`ABL_app_name`*`/sessions`service.

HTTP Operation

```
DELETE
```

URI

```
//host_name:port/oemanager/applications/App_name/requests?requestID=value&sessionID=value
```

Media type

```
NA
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X DELETE -v "http://localhost:16680/oemanager/applications/pas1
/requests?requestID=z9WHoRgGY4fnExrFWKWVlw&sessionID=G1TTNRPT3KiSOhkbyeSow"
```

Request body example

NA

Response body example

```
{
    "operation":"CANCEL ACTIVE REQUEST ON SERVER",
    "result":"",
    "outcome":"FAILURE",
    "errmsg":"Session",
    "versionStr":"PASOE 11.5.0",
    "versionNo":1
}
```

Dump a session stack

Description

Dump a process stack for a specific session.

Note: As with the classic OpenEdge Appserver, you can generate a stack for PAS for OpenEdge by using a debugging tool. For the classic Appserver, you can also run the `proGetStack` utility to generate stack traces. For PAS for OpenEdge and its multi-session agent, this REST call is analogous to `proGetStack`.

HTTP Operation

`GET`

URI

`//`*`host_name`*`:`*`port`*`/oemanager/applications/`*`App_name`*`/agents/`*`agentid`*`/sessions/`*`sessionid`*`/stacks`

Media type

`Application/vnd.progress+json`

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v -u username:password
"http://localhost:8810/oemanager/applications/oepas1/agents/P9PDwujXQqeFoGgnL7wZfw/sessions/7/stacks"
```

Request body example

NA

Response body examples

Success

```
{"outcome":"SUCCESS",
      "errmsg":"", "versionStr":"v11.7.1 ( 2017-06-09)","versionNo":1,
      "operation":"GET SESSION RUNTIME STACKS",
      "result":{"ABLStacks":[{"AgentSessionId":7, "StartupParams":"-pf
      \/usr1\/ellis\/117\/dlc\/startup.pf, -cpinternal ISO8859-1,-cpstream
ISO8859-1,-cpcoll
      Basic,-cpcase Basic, -d mdy,-numsep 44,-numdec 46,(end .pf),-logginglevel 2,
      -logfile \/usr1\/ellis\/117\/wrk\/oepas1\/logs\/oepas1.agent.log, -uburl
      AppServerDC:\/\/0.0.0.0:52511\/, -logname oepas1,-logentrytypes
      ASPlumbing,DB.Connects, -ubpropfile
      \/usr1\/ellis\/117\/wrk\/oepas1\/conf\/openedge.properties, -ASID 1,-ipver
IPv4,-sectok
      XXXXXXXXXXXXXXXXXXXXX, -T \/usr1\/ellis\/117\/wrk\/oepas1\/temp, -db
      \/usr1\/ellis\/117\/wrk\/sports2000","Propath":…(more)
```

Failure

```
{"outcome":"FAILURE",
"errmsg":"Communication layer message: General Error:READPACKET IOException :
java.io.EOFException (null). (7175)",
"versionStr":"v11.7.1( 2017-06-09 )",
"versionNo":1,"operation":"GET SESSION RUNTIME STACKS",
"result":""}
```

Terminate a session

Description

Terminate the session identified by the specified `sessionID`.

HTTP Operation

`DELETE`

URI

//host_name:*port*/oemanager/applications/*App_name*
/sessions?sessionID=*value*&terminateOpt={ 0 | 1 }

where a `terminateOpt` value of `0` causes a graceful termination and a value of `1` causes a forced termination.

Media type

NA

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X DELETE -v
"http://localhost:16680/oemanager/applications/pas1/sessions?sessionID=123&terminateOpt=1"
```

Request body example

NA

Response body example

```
{
    "operation":"TERMINATE SESSIONS",
    "result":"","outcome":
    "FAILURE","errmsg":
    "Session","versionStr":
    "PASOE 11.5.0","versionNo":1
}
```

Terminate idle ABL sessions

Description

Terminate all idle sessions associated with a specific agent.

Note: With the classic OpenEdge AppServer, you could trim idle sessions periodically by terminating agents, since each agent is a session. However, with PAS for OpenEdge and its multisession agent, trimming the agent could terminate both active and idle sessisons. Therefore, you should use this REST API call to trim the idle sessions for a specified agent.

HTTP Operation

DELETE

URI

//host_name:*port*/oemanager/applications/*App_name*
/*agentid*/sessions

Media type

Application/vnd.progress+json

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X DELETE -v -u username:password
"http://localhost:8810/oemanager/applications/oepas1/WKoQ7x5MQMKWHJoyOGp8uQ/sessions"
```

Request body example

NA

Response body examples

Success

```
{"operation":"TERMINATE SESSIONS",
"outcome":"SUCCESS","errmsg":"",
"versionStr":"v11.6.3 ( 2017-04-25 )",
"versionNo":1,
"result":5}
```

Failure

```
{"operation":"TERMINATE SESSIONS",
"outcome":"FAILURE",
"errmsg":"AgentNotFound[SessionManager.terminateFreeSessions. Error! Agent not found.
Agent Id: \"P9PDwujXQqeFoGgnL7wZfw\"]:Agent",
"versionStr":"v11.6.3 ( 2017-04-25)",
"versionNo":1,
"result":""}
```

Agent management

This section describes the REST services that are available for managing Progress Application Server for OpenEdge agents.

See also

List agents on page 256
Get agent status information on page 260
Get session metrics on page 261
Get connection information on page 262
Get request information on page 264
Get configuration properties on page 267
Update configuration properties on page 268
Stop a multi-session agent on page 270

List agents

Description

Get a list of all the PAS Agents available to an ABL application.

HTTP Operation

`GET`

URI

//host_name:*port*/oemanager/applications/*ABL_app_name*/agents

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v http://localhost:16680/oemanager/applications/pas1/agents
```

Request body example

NA

Response body example

```
{
  "result": {
    "agents": [
      {
        "agentId": "AG-sCIVXeFSQYmcb7RcHo10Zw",
        "pid": "12924",
        "state": "AVAILABLE"
      }
    ]
  },
  "operation": "GET AGENTS",
  "versionStr": "PASOE 11.5.0",
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg": ""
}
```

Check ABL object tracking status

Checks whether ABL object tracking is enabled or disabled. To change the status, see Enable or disable ABL object tracking on page 258.

HTTP Operation

GET

URI

http(s)://*host_name*:*port*/oemanager/applications/*app_name*/agents/*agentPID*/ABLObjects/status

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v -u username:password
http://localhost:16680/oemanager/applications/oepas1/18668/ABLObjects/status
```

Request body example

```
NA
```

Response body example

Result is false when tracking is off. Result is true when tracking is on. In this example ABL object tracking is disabled.

```
{
     "result": false,
     "errmsg": "",
     "versionStr": "v11.7.4 ( 2018-09-17 )",
     "versionNo": 1,
     "outcome": "SUCCESS",
     "operation": "TRACKING ABL OBJECTS"
}
```

Enable or disable ABL object tracking

Description

Enable or disable the tracking of ABL objects. When enabled, ABL objects are tracked to identify potential memory leaks. To generate a report of these objects, see Get a report of tracked ABL objects on page 259. When object tracking is not required, disable. Disabled by default.

HTTP Operation

```
PUT
```

URI

```
http(s)://host_name:port/oemanager/applications/app_name/agents/agentPID/ABLObjects/status
```

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X PUT -v -u username:password
http://localhost:16680/oemanager/applications/oepas1/18668/ABLObjects/status
```

Request body example

```
{"enable":{"true" |  "false"} }
```

Response body example

Result returns true when value is changed. To get the current value, see Check ABL object tracking status on page 257.

```
{
    "result": true,
    "errmsg": "",
    "versionStr": "v11.7.4 ( 2018-09-17 )",
    "versionNo": 1,
    "outcome": "SUCCESS",
    "operation": "TRACK ABL OBJECTS"
}
```

Get a report of tracked ABL objects

Description

Get a report of ABL objects that may indicate potential memory leaks. Getting a report requires that object tracking is enabled. See Enable or disable ABL object tracking on page 258. After leaks have been identified and resolved, disable ABL object tracking.

HTTP Operation

GET

URI

http(s)://*host_name*:*port*/oemanager/applications/*app_name*/agents/*agentPID*/ABLObjects

Media type

application/vnd.progress+json

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v -u username:password
http://localhost:16680/oemanager/applications/oepas1/18668/ABLObjects
```

Request body example

```
NA
```

Response body example

This excerpt shows the objects that are being tracked. `Source` identifies the related code.

```
{
    "result": {
        "ABLOutput": {
            "ABLObjects": [
                {
                    "AgentSessionId": 4,
                    "Objects": []
                },
                {
                    "AgentSessionId": 7,
                    "Objects": []
                }
            ]
        },
        "ABLReturnVal": true,
        "agentId": "y11SsN0cSC2skp9NV-N0Ww",
        "pid": "6064"
    },
    "errmsg": "",
    "versionStr": "v11.7.4 ( 2018-09-17 )",
    "versionNo": 1,
    "outcome": "SUCCESS",
    "operation": "GET ABL OBJECTS REPORT"
}
```

This excerpt shows results if object tracking is disabled.

```
{
    "result": {
        "ABLReturnVal": false,
        "agentId": "y11SsN0cSC2skp9NV-N0Ww",
        "pid": "6064"
    },
    "errmsg": "",
    "versionStr": "v11.7.4 ( 2018-09-17 )",
    "versionNo": 1,
    "outcome": "SUCCESS",
    "operation": "GET ABL OBJECTS REPORT"
}
```

Get agent status information

Description

Get a summary of sessions, connections and requests for an agent of an ABL application.

HTTP Operation

`GET`

URI

//host_name:*port*/oemanager/applications/*ABL_app_name*/agents/*agentID*/status

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v
http://localhost:16680/oemanager/applications/pas1/agents/AG-sCIVXeFSQYmcb7RcHo10Zw/status
```

Request body example

NA

Response body example

```
{
  "result": {
    "threads": 6,
    "sessions": 5,
    "connections": 5,
    "requests": 16
  },
  "operation": "GET AGENT STATUS",
  "versionStr": "PASOE 11.5.0",
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg": ""
}
```

Get session metrics

Description

Get session metrics for an agent of an ABL application.

HTTP Operation

GET

URI

//host_name:*port*/oemanager/applications/*ABL_app_name*/agents/*agentID*/sessions

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v
http://localhost:16680/oemanager/applications/pas1/agents/AG-sCIVXeFSQYmcb7RcHo10Zw/sessions
```

Request body example

NA

Response body example


```
{
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg": "",
  "versionStr": "v11.7.3 ( 2018-04-06 )",
  "operation": "",
  "result": {
    "AgentSession": Array [200] [
      {
        "SessionId": 4,
        "SessionState": "IDLE",
        "StartTime": "2018-04-13T11:14:12.865",
        "EndTime": null,
        "ThreadId": -1,
        "ConnectiondId": null,
        "SessionExternalState": 0,
        "SessionMemory": 6108819
      },
      {
        "SessionId": 6,
        "SessionState": "ACTIVE",
        "StartTime": "2018-04-13T11:14:12.865",
        "EndTime": null,
        "ThreadId": 125,
        "ConnectiondId": null,
        "SessionExternalState": 0,
        "SessionMemory": 6310022
      },
      {
        "SessionId": 7,
        "SessionState": "IDLE",
        "StartTime": "2018-04-13T11:14:12.865"",
        "EndTime": null,
        "ThreadId": -1,
        "ConnectiondId": null,
        "SessionExternalState": 0,
        "SessionMemory": 6105987
      },
    ]
```


Get connection information

Description

Get information on the connections of an agent of an ABL application.

HTTP Operation

GET

URI

//host_name:*port*/oemanager/applications/*ABL_app_name*/agents/*agentID*/connections

Media type

application/vnd.progress+json

Response codes

200 Success
500 Unexpected Server Error

Command-line example

```
curl -X GET -v
http://localhost:16680/oemanager/applications/pas1/agents/AG-sCIVXeFSQYmcb7RcHo10Zw/connections
```

Request body example

NA

Response body example


```
{
  "result": {
    "AgentConnection": [
      {
        "ConnectionId": 7,
        "ConnectionState": "LISTENER",
        "SessionId": -1
      },
      {
        "ConnectionId": 10,
        "ConnectionState": "LISTENER",
        "SessionId": -1
      },
      {
        "ConnectionId": 11,
        "ConnectionState": "ADMINISTRATOR",
        "SessionId": -1
      },
      {
        "ConnectionId": 19,
        "ConnectionState": "ADMINISTRATOR",
        "SessionId": -1
      },
      {
        "ConnectionId": 22,
        "ConnectionState": "REQUEST",
        "SessionId": -1
      }
    ]
  },
  "operation": "GET CONNECTIONS METRICS",
  "versionStr": "PASOE 11.5.0",
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg": ""
}
```

Get request information

Description

Get request information for an agent of an ABL application.

HTTP Operation

GET

URI

//host_name:*port*/oemanager/applications/*ABL_app_name*/agents/*agentID*/requests

Media type

application/vnd.progress+json

Response codes

200 Success
500 Unexpected Server Error

Command-line example

```
curl -X GET -v
http://localhost:16680/oemanager/applications/pas1/agents/AG-sCIVXeFSQYmcb7RcHo10Zw/requests
```

Request body example

NA

Response body example


```
{
  "result": {
    "AgentRequest": [
      {
        "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentThreadData",
        "SessionId": 3,
        "ConnectionId": 19,
        "StartTime": "2014-09-22T11:07:49.744",
        "EndTime": "2014-09-22T11:07:49.745",
        "RequestNum": 0
      },
      {
        "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentSessionData",
        "SessionId": 3,
        "ConnectionId": 19,
        "StartTime": "2014-09-22T11:07:49.756",
        "EndTime": "2014-09-22T11:07:49.757",
        "RequestNum": 1
      },
      {
        "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentConnectionData",
        "SessionId": 3,
        "ConnectionId": 19,
        "StartTime": "2014-09-22T11:07:49.759",
```

```
  "EndTime": "2014-09-22T11:07:49.760",
  "RequestNum": 2
},
{
  "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentRequestData",
  "SessionId": 3,
  "ConnectionId": 19,
  "StartTime": "2014-09-22T11:07:49.762",
  "EndTime": "2014-09-22T11:07:49.763",
  "RequestNum": 3
},
{
  "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentThreadData",
  "SessionId": 3,
  "ConnectionId": 19,
  "StartTime": "2014-09-22T11:07:55.812",
  "EndTime": "2014-09-22T11:07:55.813",
  "RequestNum": 4
},
{
  "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentSessionData",
  "SessionId": 3,
  "ConnectionId": 19,
  "StartTime": "2014-09-22T11:07:55.814",
  "EndTime": "2014-09-22T11:07:55.815",
  "RequestNum": 5
},
{
  "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentConnectionData",
  "SessionId": 3,
  "ConnectionId": 19,
  "StartTime": "2014-09-22T11:07:55.817",
  "EndTime": "2014-09-22T11:07:55.817",
  "RequestNum": 6
},
{
  "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentRequestData",
  "SessionId": 3,
  "ConnectionId": 19,
  "StartTime": "2014-09-22T11:07:55.819",
  "EndTime": "2014-09-22T11:07:55.819",
  "RequestNum": 7
},
{
  "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentThreadData",
  "SessionId": 3,
  "ConnectionId": 19,
  "StartTime": "2014-09-22T11:09:22.652",
  "EndTime": "2014-09-22T11:09:22.653",
  "RequestNum": 8
},
{
  "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentSessionData",
  "SessionId": 3,
  "ConnectionId": 19,
  "StartTime": "2014-09-22T11:09:22.654",
  "EndTime": "2014-09-22T11:09:22.655",
  "RequestNum": 9
},
{
  "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentConnectionData",
  "SessionId": 3,
  "ConnectionId": 19,
  "StartTime": "2014-09-22T11:09:22.657",
  "EndTime": "2014-09-22T11:09:22.657",
  "RequestNum": 10
},
{
  "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentRequestData",
```

```
    "SessionId": 3,
    "ConnectionId": 19,
    "StartTime": "2014-09-22T11:09:22.659",
    "EndTime": "2014-09-22T11:09:22.660",
    "RequestNum": 11
  },
  {
    "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentThreadData",
    "SessionId": 3,
    "ConnectionId": 19,
    "StartTime": "2014-09-22T11:09:23.001",
    "EndTime": "2014-09-22T11:09:23.002",
    "RequestNum": 12
  },
  {
    "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentSessionData",
    "SessionId": 3,
    "ConnectionId": 19,
    "StartTime": "2014-09-22T11:09:23.004",
    "EndTime": "2014-09-22T11:09:23.005",
    "RequestNum": 13
  },
  {
    "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentConnectionData",
    "SessionId": 3,
    "ConnectionId": 19,
    "StartTime": "2014-09-22T11:09:23.007",
    "EndTime": "2014-09-22T11:09:23.007",
    "RequestNum": 14
  },
  {
    "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentRequestData",
    "SessionId": 3,
    "ConnectionId": 19,
    "StartTime": "2014-09-22T11:09:23.024",
    "EndTime": "2014-09-22T11:09:23.025",
    "RequestNum": 15
  },
  {
    "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentSessionData",
    "SessionId": 3,
    "ConnectionId": 19,
    "StartTime": "2014-09-22T11:10:23.689",
    "EndTime": "2014-09-22T11:10:23.690",
    "RequestNum": 16
  },
  {
    "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentSessionData",
    "SessionId": 3,
    "ConnectionId": 19,
    "StartTime": "2014-09-22T11:10:34.078",
    "EndTime": "2014-09-22T11:10:34.079",
    "RequestNum": 17
  },
  {
    "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentSessionData",
    "SessionId": 3,
    "ConnectionId": 19,
    "StartTime": "2014-09-22T11:10:34.389",
    "EndTime": "2014-09-22T11:10:34.390",
    "RequestNum": 18
  },
  {
    "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentConnectionData",
    "SessionId": 3,
    "ConnectionId": 19,
    "StartTime": "2014-09-22T11:12:03.324",
    "EndTime": "2014-09-22T11:12:03.324",
    "RequestNum": 19
```

```
        },
        {
          "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentConnectionData",
          "SessionId": 3,
          "ConnectionId": 19,
          "StartTime": "2014-09-22T11:12:44.784",
          "EndTime": "2014-09-22T11:12:44.785",
          "RequestNum": 20
        },
        {
          "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentConnectionData",
          "SessionId": 3,
          "ConnectionId": 19,
          "StartTime": "2014-09-22T11:12:45.131",
          "EndTime": "2014-09-22T11:12:45.132",
          "RequestNum": 21
        },
        {
          "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentRequestData",
          "SessionId": 3,
          "ConnectionId": 19,
          "StartTime": "2014-09-22T11:13:13.439",
          "EndTime": "2014-09-22T11:13:13.440",
          "RequestNum": 22
        },
        {
          "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentRequestData",
          "SessionId": 3,
          "ConnectionId": 19,
          "StartTime": "2014-09-22T11:14:49.335",
          "EndTime": "2014-09-22T11:14:49.336",
          "RequestNum": 23
        },
        {
          "RequestProcName": "OpenEdge.Server.Admin.AgentInfo:getAgentRequestData",
          "SessionId": 3,
          "ConnectionId": 19,
          "StartTime": "2014-09-22T11:14:49.690",
          "EndTime": null,
          "RequestNum": 24
        }
      ]
    },
    "operation": "GET REQUESTS METRICS",
    "versionStr": "PASOE 11.5.0",
    "versionNo": 1,
    "outcome": "SUCCESS",
    "errmsg": ""
}
```


Get configuration properties

Description

Get configuration information (properties) of agents for an ABL application.

HTTP Operation

GET

URI

//host_name:*port*/oemanager/applications/*App_name*/agents/properties

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v http://localhost:16680/oemanager/applications/pas1/agents/properties
```

Request body example

NA

Response body example


```
{
  "result": {
    "workDir": "/wrk/pas1/work",
    "agentMinPort": "62002",
    "infoVersion": "9010",
    "sessionDisconnProc": "",
    "applications": "pas1",
    "PROPATH": "/wrk/pas1/openedge,/dlc/tty",
    "collectStatsData": "0",
    "agentStartupProcParam": "",
    "agentShutdownProc": "",
    "sessionShutdownProc": "",
    "collectMetrics": "1",
    "sessionStartupProcParam": "",
    "flushStatsData": "0",
    "sessionDeactivateProc": "",
    "sessionActivateProc": "",
    "allowRuntimeUpdates": "0",
    "sessionConnectProc": "",
    "sessionTimeout": "180",
    "sessionStartupProc": "",
    "statusEnabled": "1",
    "agentMaxPort": "62202",
    "uuid": "http://localhost:9999/oepas1",
    "agentStartupProc": ""
  },
  "operation": "GET AGENT PROPERTIES",
  "versionStr": "PASOE 11.5.0",
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg": ""
}
```


Update configuration properties

Description

Update the configuration properties in `conf/openedge.properties` for agents for an ABL application.

HTTP Operation

PUT

URI

//host_name:*port*/oemanager/applications/*App_name*/agents/properties
{"*prop_name*"=*value* **[**, "*prop_name*"=*value* **[**, **. . .]] }**

"*prop_name*"=*value* **[**, *prop_name*"=*value* **[**, **. . .]]**

Set one or more properties expressed as name/value pairs in a comma-separated list. See the /conf/openedge.properties.README file for a description of properties and their possible values.

Media type

application/vnd.progress+json

Response codes

200 Success
500 Unexpected Server Error

Command-line example

```
curl -X PUT -v http://localhost:16680/oemanager/applications/pas1/agents/properties
-d '{"collectMetrics":0}' -H "Content-Type: application/vnd.progress+json"
```

Request body example

```
{"collectMetrics":0}
```

Response body example

NA

Stop a multi-session agent

Description

Stop a multi-session agent that belongs to an ABL application. Note that you identify the agent by the *agentID*, which you can obtain by executing /oemanager/applications/*ABL_app_name*/agents. You can specify, in milliseconds, how long the stop process must wait before stopping an agent, if the agent is currently serving a request, using the waitToFinish parameter. waitToFinish stops new requests to the agent and waits for any current requests to complete. If the current requests do not complete in the specified time, a STOP message is sent to the agent. You can also specify an additional wait time, in milliseconds, using the waitAfterStop parameter. waitAfterStop checks if the agent has indeed stopped. If the agent has not stopped, then at the end of the specified time, a KILL message is sent to the agent. In either case, if the agent finishes processing its current requests before the waitToFinish time expires, the agent will immediately shut down.

Note: Both waitToFinish and waitAfterStop are optional parameters. If you do not specify these, the agent will be stopped immediately, irrespective of whether the agent is currently idle or serving a request.

HTTP Operation

```
DELETE
```

URI

```
//host_name:port/oemanager/applications/App_name/agents/agentID?
waitToFinish=time-in-milliseconds&waitAfterStop=time-in-milliseconds
```

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
403 Access Denied
500 Unexpected Server Error
```

Command-line example

```
curl -X DELETE -v
http://localhost:16680/oemanager/applications/oepas1/agents/AG-sCIVXeFSQYmcb7RcHo10Zw?waitToFinish=
60000 -u tomcat:tomcat -H "Content-Type: application/vnd.progress+json"
```

Request body example

NA

Response body example

```
{
  "result": {
    "agentID": "AG-sCIVXeFSQYmcb7RcHo10Zw"
  },
  "versionStr": "v11.7.3 ( 2018-02-12 )",
```

```
  "versionNo": 1,
  "outcome":"SUCCESS",
  "errmsg":"",
  "operation": "STOP AGENT ",
  }
```

Add a multi-session agent

Description

Add a multi-session agent to an ABL application.

HTTP Operation

POST

URI

//host_name:*port*/oemanager/applications/*App_name*/addAgent

Media type

application/vnd.progress+json

Response codes

```
200 Success
403 Access Denied
500 Unexpected Server Error
```

Command-line example

```
curl -X POST -v http://localhost:16680/oemanager/applications/oepas1/addAgent
```

Request body example

NA

Response body example

```
{
  "result": {
  "agentID": "9RT0PpxRTk6qPS1XEA_QSg", "pid":"84708","state":"AVAILABLE"
  },
  "versionStr": "v11.7.3 ( 2018-02-12 )",
  "versionNo": 1,
  "outcome":"SUCCESS",
  "errmsg":"",
  "operation": "GET ABL OBJECTS REPORT",
  }
```

Transport management

This section describes the REST services that are available for managing PAS for OpenEdge SOAP, REST, and APSV transports.

See also

Get transport configuration properties (SOAP/REST) on page 272
Update properties of a transport (SOAP/REST) on page 274
Enable or disable a transport (SOAP/REST) on page 275
Enable or disable a transport (APSV) on page 276
Get runtime metrics of a transport (SOAP/REST) on page 277
Get runtime metrics of a transport (APSV) on page 279
Reset runtime metrics for a transport (SOAP/REST) on page 280
Reset runtime metrics of a transport (APSV) on page 281
List deployed services (SOAP/REST) on page 282
Get information about a service (SOAP/REST) on page 283
Get configuration properties of a service (SOAP) on page 285
Deploy a new service (REST) on page 286
Deploy a new service (SOAP) on page 287
Update runtime properties of a service (SOAP) on page 288
Get runtime metrics for a service (SOAP) on page 289
Reset runtime metrics for a service (SOAP) on page 290
Enable or disable a service (SOAP) on page 291
Remove a service (SOAP/REST) on page 292

Get transport configuration properties (SOAP/REST)

Description

Get configuration properties of a REST or SOAP transport of a Web application belonging to an ABL application.

HTTP Operation

`GET`

URI

//host_name:*port*/oemanager/applications/
ABL_app_name/webapps/*Web_app_name*/transports/ { soap | rest } /properties

Media type

`application/vnd.progress+json`

Request body

`NA`

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example (SOAP)

```
curl -X GET -v
http://localhost:16680/oemanager/server/oeservices/ROOT/transports/soap/properties
```

Response body example (SOAP)

```
{"result":
          {"idleSessionTimeout":0,
           "waitIfBusy":0,
           "connectionLifetime":0,
           "requestWaitTimeout":-1,
           "staleO4GLObjectTimeout":0,
           "clientASKActivityTimeout":60,
           "initialSessions":1,
           "clientASKResponseTimeout":60,
           "minIdleConnections":0,
           "maxSessions":0,
           "minSessions":1,
           "appServerKeepalive":"denyClientASK,allowServerASK",
           "serviceFaultLevel":2
          },
 "outcome":"SUCCESS",
 "versionNo":1,
 "errmsg":"NA",
 "versionStr":
 "PASOE 11.5.0",
 "operation":"GET SOAP TRANSPORT PROPERTIES"
}
```

Command-line example (REST)

```
curl -X GET -v
http://localhost:16680/oemanager/server/oeservices/ROOT/transports/rest/properties
```

Response body example (REST)

```
{
  "result": {
    "appServerKeepalive": "denyClientASK,allowServerASK",
    "idleSessionTimeout": 0,
    "minSessions": 1,
    "requestWaitTimeout": -1,
    "initialSessions": 1,
    "waitIfBusy": 0,
    "staleO4GLObjectTimeout": 0,
    "minIdleConnections": 0,
    "connectionLifetime": 0,
    "serviceFaultLevel": 2,
    "clientASKResponseTimeout": 60,
    "maxSessions": 0,
    "clientASKActivityTimeout": 60
  },
  "operation": "GET REST TRANSPORT PROPERTIES",
  "versionStr": "PASOE 11.5.0",
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg": "
```

Update properties of a transport (SOAP/REST)

Description

Update the properties of a SOAP or REST transport of a Web application belonging to an ABL application.

HTTP Operation

```
PUT
```

URI

```
//host_name:port/oemanager/applications/
ABL_app_name/webapps/Web_app_name/transports/{soap|rest}/properties
```

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example (SOAP)

```
curl -X PUT -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/soap/properties
-H "Content-Type: application/vnd.progress+json" -d '{"waitIfBusy":1}'
```

Request body example (SOAP)

```
{"waitIfBusy":1}
```

Response body example (SOAP)

```
{"result":
          {"idleSessionTimeout":0,
           "waitIfBusy":1,
           "connectionLifetime":0,
           "requestWaitTimeout":-1,
           "staleO4GLObjectTimeout":0,
           "clientASKActivityTimeout":60,
           "initialSessions":1,
           "clientASKResponseTimeout":60,
           "minIdleConnections":0,
           "maxSessions":0,
           "minSessions":1,
           "appServerKeepalive":"denyClientASK,allowServerASK",
           "serviceFaultLevel":2
          },
 "outcome":"SUCCESS",
 "versionNo":1,
 "errmsg":"NA",
 "versionStr":
 "PASOE 11.5.0",
```

```
 "operation":"SET SOAP TRANSPORT PROPERTIES"
}
```

Command-line example (REST)

```
curl -X PUT -v
http://localhost:16680/oemanager/server/oeservices/ROOT/transports/rest/properties
-H "Content-Type: application/vnd.progress+json" -d '{"maxSessions":1}'
```

Request body example (REST)

```
{"maxSessions":1}
```

Response body example (REST)

```
{
 "operation":"SET REST TRANSPORT PROPERTIES",
 "result":{"appServerKeepalive":"denyClientASK,allowServerASK",
           "idleSessionTimeout":0,
           "minSessions":1,
           "requestWaitTimeout":-1,
           "initialSessions":1,
           "waitIfBusy":0,
           "staleO4GLObjectTimeout":0,
           "minIdleConnections":0,
           "connectionLifetime":0,
           "serviceFaultLevel":2,
           "clientASKResponseTimeout":60,
           "maxSessions":1,
           "clientASKActivityTimeout":60
          },
 "outcome":"SUCCESS",
 "errmsg":"NA",
 "versionStr":"PASOE 11.5.0",
 "versionNo":1
}
```

Enable or disable a transport (SOAP/REST)

Description

Enable or disable the REST or SOAP transport of a Web application belonging to an ABL application.

HTTP Operation

PUT

URI

//host_name:*port*/oemanager/applications/
ABL_app_name/webapps/*Web_app_name*/transports/ { soap | rest } /state

Media type

application/vnd.progress+json

Request body

```
{
  "STATE" : "{ENABLED | DISABLED}"
}
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example (SOAP)

```
curl  -X PUT -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/soap/state
-d '{"state":"ENABLED"}'  -H "Content-Type: application/vnd.progress+json"
```

Response body example (SOAP)

```
{"state":"ENABLED"}
```

Command-line example (REST)

```
curl  -X PUT -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/rest/state
-d '{"state":"DISABLED"}'  -H "Content-Type: application/vnd.progress+json"
```

Response body example (REST)

```
{"state":"DISABLED"}
```

Enable or disable a transport (APSV)

Description

Enable or disable the APSV transport of a Web application belonging to an ABL application.

HTTP Operation

```
PUT
```

URI

```
//host_name:port/oemanager/applications/ABL_app_name
/webapps/Web_app_name/transports/state
```

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X DELETE  -d '{"state":"DISABLED"}'  -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/state
```

Request body example

```
{"state":"DISABLED"}
```

Response body example

```
{
 "operation":"SET APSV TRANSPORT STATE",
 "result":"NA",
 "versionNo":1,
 "versionStr":"PASOE 11.5.0",
 "outcome":"SUCCESS",
 "errmsg":"NA"
}
```

Get runtime metrics of a transport (SOAP/REST)

Description

Get runtime metrics for a REST or SOAP transport of a Web application deployed in an ABL application.

HTTP Operation

GET

URI

//host_name:*port*/oemanager/applications/
ABL_app_name/webapps/*Web_app_name*/transports/ { soap | rest } /metrics

Media type

```
application/vnd.progress+json
```

Request body

```
NA
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example (SOAP)

```
curl -x GET -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/soap/metrics
```

Response body example (SOAP)

```
{"result":
         {
         "serviceDisabled":0,
         "soapProcessorErrors":0,
         "httpRequestErrors":0,
         "httpRequests":0,
         "statusRequest":0,
         "methodNotAllowederrors":0,
         "soapRequests":0,
         "successfulSoapRequests":0,
         "wsdlRequest":0,
         "activeRequests":0,
         "urlNotFoundErrors":0,
         "type":"OE_SOAP_TRANSPORT",
         "startTime":"2014-07-17T08:40:03.422-04:00",
         "accessTime":"2014-07-17T10:34:00.295-04:00"
         },
 "outcome":"SUCCESS",
 "versionNo":1,
 "errmsg":"NA",
 "versionStr":"PASOE 11.5.0",
 "operation":"GET OE_SOAP_TRANSPORT METRICS"
}
```

Command-line example (REST)

```
curl -X GET -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/rest/metrics
```

Response body example (REST)

```
{
 "result":{"successfulConnectRequests":0,
           "serviceNotFound":0,
           "numRequests":0,
           "serviceUnavailableRequests":0,
           "statusRequests":0,
           "runRequest":0,
           "connectRequests":0,
           "successfulRequests":0,
           "ExpressionError":0,
           "failedRequests":0,
           "successfulRunRequest":0,
           "type":"OE_REST_TRANSPORT",
           "accessTime":"2014-07-17T08:40:58.315-04:00"
 },
 "outcome":"SUCCESS",
 "versionNo":1,
 "errmsg":"NA",
 "versionStr":"PASOE 11.5.0",
 "operation":"GET OE_REST_TRANSPORT METRICS"
 }
```

Get runtime metrics of a transport (APSV)

Description

Get the runtime metrics for the APSV transport for a Web application deployed in an ABL application.

HTTP Operation

```
GET
```

URI

```
//host_name:port/oemanager/applications/ABL_app_name
/webapps/Web_app_name/transports/apsv/metrics
```

Media type

```
application/vnd.progress+json
```

Request body

NA

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/apsv/metrics
```

Response body example

```
{
"operation":"GET OE_APSV_TRANSPORT METRICS",
"result":{"sessionErrors":0,
          "sessionRequest":0,
          "disconnectRequest":0,
          "connectErrors":0,
          "connectRequest":0,
          "statusRequest":0,
          "disconnectErrors":0,
          "forbiddenErrors":0,
          "type":"OE_APSV_TRANSPORT",
          "accessTime":"2014-07-18T04:34:40.743-04:00",
          "lastResetTime":"2014-07-18T04:34:22.314-04:00"
         },
 "versionNo":1,
 "versionStr":"PASOE 11.5.0",
 "outcome":"SUCCESS",
 "errmsg":"NA"
}
```

Reset runtime metrics for a transport (SOAP/REST)

Description

Reset aggregate runtime metrics for a SOAP or REST transport of a Web application deployed in an ABL application.

HTTP Operation

```
DELETE
```

URI

```
//host_name:port/oemanager/applications/
ABL_app_name/webapps/Web_app_name/transports/{soap|rest}/metrics
```

Media type

```
application/vnd.progress+json
```

Request body

NA

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example (SOAP)

```
curl -X DELETE -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/rest/metrics
```

Response body example (SOAP)

```
{
 "result":"NA",
 "outcome":"SUCCESS",
 "versionNo":1,
 "errmsg":"NA",
 "versionStr":"PASOE 11.5.0",
 "operation":"RESET OE_REST_TRANSPORT METRICS"
}
```

Command-line example (REST)

```
curl -X DELETE -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/soap/metrics
```

Response body example (REST)

```
{
 "result":"NA",
```

```
 "outcome":"SUCCESS",
 "versionNo":1,
 "errmsg":"NA",
 "versionStr":"PASOE 11.5.0",
 "operation":"RESET OE_SOAP_TRANSPORT METRICS"
}
```

Reset runtime metrics of a transport (APSV)

Description

Reset aggregate runtime metrics for an APSV transport of a Web application deployed in an ABL application.

HTTP Operation

```
DELETE
```

URI

```
//host_name:port/oemanager/applications/ABL_app_name
/webapps/Web_app_name/transports/metrics
```

Media type

```
application/vnd.progress+json
```

Request body

NA

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X DELETE -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/metrics
```

Response body example

```
{
 "result":"NA",
 "outcome":"SUCCESS",
 "versionNo":1,
 "errmsg":"NA",
 "versionStr":"PASOE 11.5.0",
 "operation":"RESET OE_APSV_TRANSPORT METRICS"
}
```

List deployed services (SOAP/REST)

Description

Get a list all the deployed SOAP or REST services in a Web application deployed in an ABL application.

HTTP Operation

```
GET
```

URI

```
//host_name:port/oemanager/applications/App_name/webapps/Web_app_name/transports/{soap|rest}/oeservices
```

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Request body

NA

Command-line example (SOAP)

```
curl -X GET -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/soap/oeservices
```

Response body example(SOAP)

```
{
  "result": {
    "OESoapServiceDesc": [
      {
        "archiveLocation": "/webapps/ROOT/WEB-INF/adapters/soap/ROOT/SimpleTest.wsdl",
        "archiveName": "SimpleTest.wsm",
        "accessUrl": "urn:nxgas:simpletestsf",
        "oetype": "SOAP_DESCRIPTOR",
        "version": "11.5.0",
        "description": "PAS OpenEdge SOAP Service Artifact",
        "name": "SimpleTest",
        "state": "ENABLED",
        "type": "OPENEDGE"
      }
    ]
  },
  "operation": "GET SOAP TRANSPORT DESCRIPTORS",
  "versionStr": "PASOE 11.5.0",
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg": ""
}
```

Command-line example (REST)

```
curl -X GET -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/rest/oeservices
```

Response body example(REST)

```
{
  "result": {
    "OERestServiceDesc": [
      {
        "archiveLocation":
"/webapps/ROOT/WEB-INF/adapters/rest/_oepingService/_oepingService.paar",
        "archiveName": "_oepingService.paar",
        "accessUrl": "/rest/_oepingService",
        "oetype": "REST_DESCRIPTOR",
        "version": "11.5.0",
        "description": "PAS OpenEdge REST Service Descriptor",
        "name": "_oepingService",
        "type": "OPENEDGE",
        "uri": "http://localhost:16680/rest/_oepingService"
      }
    ]
  },
  "operation": "GET REST TRANSPORT DESCRIPTORS",
  "versionStr": "PASOE 11.5.0",
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg": ""
}
```

Get information about a service (SOAP/REST)

Description

Get details about a named SOAP or REST service in a Web application deployed in an ABL application.

HTTP Operation

GET

URI

//host_name:*port*/oemanager/applications/*ABL_app_name*
/webapps/*Web_app_name*/transports/{soap|rest}/oeservices/{*SoapServiceName*|*RestSeviceName*}

Media type

application/vnd.progress+json

Response codes

200 Success
500 Unexpected Server Error

Request body

NA

Command-line example (SOAP)

```
curl -X GET -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/soap/oeservices/SimpleTest
```

Response body example (SOAP)


```
{
  "result": {
    "OESoapServiceDesc": [
      {
        "archiveLocation": "/WEB-INF/adapters/soap/ROOT/",
        "archiveName": "SimpleTest.wsm",
        "accessUrl": "urn:nxgas:simpletestsf",
        "oetype": "SOAP_DESCRIPTOR",
        "version": "11.5.0",
        "description": "PAS OpenEdge SOAP Service Artifact",
        "name": "SimpleTest",
        "state": "ENABLED",
        "type": "OPENEDGE"
      }
    ]
  },
  "operation": "GET SOAP TRANSPORT DESCRIPTOR",
  "versionStr": "PASOE 11.5.0",
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg": ""
}
```


Command-line example (REST)

```
curl -X GET -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/rest/oeservices/_oepingService
```

Response body example (REST)


```
{
  "result": {
    "OERestServiceDesc": [
      {
        "archiveLocation": "/webapps/ROOT/WEB-INF/adapters/rest/_oepingService/",
        "archiveName": "_oepingService.paar",
        "accessUrl": "/rest/_oepingService",
        "oetype": "REST_DESCRIPTOR",
        "version": "11.5.0",
        "description": "PAS OpenEdge REST Service Descriptor",
        "name": "_oepingService",
        "type": "OPENEDGE",
        "uri": "http://localhost:16680/rest/_oepingService"
      }
    ]
  },
  "operation": "GET REST TRANSPORT DESCRIPTOR",
  "versionStr": "PASOE 11.5.0",
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg": ""
}
```

Get configuration properties of a service (SOAP)

Description

Get configuration properties of a named SOAP service in a Web application deployed in an ABL application.

HTTP Operation

```
GET
```

URI

```
//host_name:port/oemanager/applications/ABL_app_name/webapps/Web_app_name
/transports/soap/oeservices/SoapServiceName/properties
```

Media type

```
application/vnd.progress+json
```

Response body

```
NA
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/soap/oeservices/SimpleTest/properties
```

Request body example

```
{"result":
          {"idleSessionTimeout":0,
           "waitIfBusy":0,
           "connectionLifetime":0,
           "requestWaitTimeout":-1,
           "staleO4GLObjectTimeout":0,
           "clientASKActivityTimeout":60,
           "initialSessions":1,
           "clientASKResponseTimeout":60,
           "minIdleConnections":0,
           "maxSessions":0,
           "minSessions":1,
           "appServerKeepalive":"denyClientASK,allowServerASK",
           "serviceFaultLevel":2
          },
 "outcome":"SUCCESS",
 "versionNo":1,
 "errmsg":"NA",
 "versionStr":
 "PASOE 11.5.0",
```

```
 "operation":"GET SOAP TRANSPORT PROPERTIES"
}
```

Deploy a new service (REST)

Description

Deploy a new REST service in a WebApp deployed in an ABL application.

HTTP Operation

POST

URI

//host_name:*port*/oemanager/applications/*ABL_app_name*/webapps
/*Web_app_name*/transports/rest/oeservices/

Media type

application/vnd.progress.paar+zip

Content disposition

attachment; filename={ *AppNewService*.paar | *AppNewService*.zip }

Response codes

200 Success
500 Unexpected Server Error

Command-line example

```
curl -X POST -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/rest/oeservices
-data-binary @PingService1.paar
-H "Accept: application/vnd.progress+json"
-H "Content-Type:application/vnd.progress.paar+zip"
-H "Content-Disposition: attachment; filename=PingService1.paar"
```

Request body example

application/vnd.progress.paar+zip

Response body example

```
{
  "result": {
    "OERestServiceDesc": [
      {
        "archiveLocation": "WEB-INF/adapters/rest/PingService1/PingService1.paar",
        "archiveName": "PingService1.paar",
        "accessUrl": "/rest/PingService1",
        "oetype": "REST_DESCRIPTOR",
```

```
        "version": "11.5.0",
        "description": "PAS OpenEdge REST Service Descriptor",
        "name": "PingService1",
        "type": "OPENEDGE",
        "uri": "http://localhost:16680/rest/PingService1"
      }
    ]
  },
  "operation": "DEPLOY REST TRANSPORT DESCRIPTOR",
  "versionStr": "PASOE 11.5.0",
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg"
```

Deploy a new service (SOAP)

Description

Deploy a new SOAP service in a WebApp deployed in an ABL application.

HTTP Operation

POST

URI

```
//host_name:port/oemanager/applications/ABL_app_name/webapps
/Web_app_name/transports/soap/oeservices/
```

Media type

```
application/vnd.progress.wsm+xml
```

Content disposition

```
attachment; filename=AppNewService.wsm
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl  -X POST  -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/soap/oeservices
-data-binary @SimpleTest2.wsm
-H "Accept: application/vnd.progress+json"
-H "Content-Type: application/vnd.progress.wsm+xml"
-H "Content-Disposition: attachment; filename=SimpleTest2.wsm"
```

Request body example

```
application/vnd.progress.wsm+xm
```

Response body example

```
{
  "result": {
    "OESoapServiceDesc": [
      {
        "archiveLocation": "WEB-INF/adapters/soap/ROOT/SimpleTest.wsdl",
        "archiveName": "SimpleTest.wsm",
        "accessUrl": "urn:nxgas:simpletestsf",
        "oetype": "SOAP_DESCRIPTOR",
        "version": "11.5.0",
        "description": "PAS OpenEdge SOAP Service Artifact",
        "name": "SimpleTest",
        "state": "ENABLED",
        "type": "OPENEDGE"
      }
    ]
  },
  "operation": "DEPLOY SOAP TRANSPORT DESCRIPTOR",
  "versionStr": "PASOE 11.5.0",
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg": ""
}
```

Update runtime properties of a service (SOAP)

Description

Update the runtime properties of a SOAP service in the Web application of a deployed ABL application.

HTTP Operation

PUT

URI

//host_name:*port*/oemanager/applications/*ABL_app_name*
/webapps/*Web_app_name*/transports/soap/oeservices/*SoapServiceName*/properties

Media type

application/vnd.progress+json

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X PUT -v
http://oelxdev03.bedford.progress.com:16680/oemanager/server/oeservices/ROOT/transports/soap/service/TestService2/properties
-H "Content-Type: application/vnd.progress+json" -d '{"maxSessions":1}'
```

Request body example

```
{"maxSessions":1}
```

Response body example

```
{"result":
          {"idleSessionTimeout":0,
           "waitIfBusy":1,
           "connectionLifetime":0,
           "requestWaitTimeout":-1,
           "staleO4GLObjectTimeout":0,
           "clientASKActivityTimeout":60,
           "initialSessions":1,
           "clientASKResponseTimeout":60,
           "minIdleConnections":0,
           "maxSessions":1,
           "minSessions":1,
           "appServerKeepalive":"denyClientASK,allowServerASK",
           "serviceFaultLevel":2
          },
 "outcome":"SUCCESS",
 "versionNo":1,
 "errmsg":"NA",
 "versionStr":
 "PASOE 11.5.0",
 "operation":"SET SOAP TRANSPORT PROPERTIES"
}
```

Get runtime metrics for a service (SOAP)

Description

Update the runtime metrics of a SOAP service in the Web application deployed in an ABL application.

HTTP Operation

GET

URI

```
//host_name:port/oemanager/applications/ABL_app_name
/webapps/Web_app_name/transports/soap/oeservices/SoapServiceName/metrics
```

Media type

```
application/vnd.progress+json
```

Request body

NA

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/soap/oeservices/SimpleTest/metrics
```

Response body example


```
{"result":
         {"requests":0,
          "appserverConnections":0,
          "nameserverError":0,
          "activeRequests":0,
          "providerError":0,
          "subAppObjects":0,
          "objPoolFull":0,
          "objPoolExpired":0,
          "connectError":0,
          "faults":0,
          "ABLClientError":0,
          "procObjects":0,
          "startTime":"2014-07-17T08:40:07.315-04:00",
          "appObjects":0,
          "objNotFound":0,
          "ABLServerError":0,
          "accessTime":"2014-07-18T02:27:07.212-04:00",
          "ABLAppError":0,
          "clientError":0,
          "type":"OE_SOAP_TRANSPORT_APP"
          },
 "outcome":"SUCCESS",
 "versionNo":1,
 "errmsg":"NA",
 "versionStr":"PASOE 11.5.0",
 "operation":"GET OE_SOAP_TRANSPORT_APP METRICS"
}
```


Reset runtime metrics for a service (SOAP)

Description

Reset runtime metrics for a SOAP service in a Web application deployed in an ABL application.

HTTP Operation

DELETE

URI

//host_name:*port*/oemanager/applications/*ABL_app_name*
/webapps/*Web_app_name*/transports/soap/oeservices/*SoapServiceName*/metrics

Media type

application/vnd.progress+json

Request body

NA

Response codes

200 Success
500 Unexpected Server Error

Command-line example

```
curl -X DELETE -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/soap/oeservices/SimpleTest/metrics
```

Response body example

```
{
 "result":"NA",
 "outcome":"SUCCESS",
 "versionNo":1,
 "errmsg":"NA",
 "versionStr":"PASOE 11.5.0",
 "operation":"RESET OE_SOAP_TRANSPORT_APP METRICS"
}
```

Enable or disable a service (SOAP)

Description

Enable or disable a SOAP service in a Web application deployed in an ABL application.

HTTP Operation

PUT

URI

```
//host_name:port/oemanager/applications/ABL_app_name/webapps
/Web_app_name/transports/soap/oeservices/SoapServiceName/state
```

Media type

```
application/vnd.progress+json
```

Request body

```
{
  "STATE" : "{ENABLED | DISABLED}"
}
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X PUT -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/soap/oeservices/SimpleTest/state
-H "Content-Type: application/vnd.progress+json" -d '{"state":"DISABLED"}'
```

Response body example

```
{"state":"DISABLED"}
```

Remove a service (SOAP/REST)

Description

Remove a specified SOAP or REST service from a Web application deployed in an ABL application.

HTTP Operation

```
DELETE
```

URI

//host_name:port/oemanager/applications/ABL_app_name/webapps/Web_app_name/transports/{soap|rest}*/oeservices*{*SoapServiceName*|*RestSeviceName*}

Media type

```
application/vnd.progress+json
```

Response codes

```
500 Unexpected Server Error
```

Request body

NA

Command-line example (SOAP)

```
curl -X DELETE -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/soap/oeservices/SimpleTest
```

Response body example (SOAP)

```
{
  "result": {
    "OESoapServiceDesc": [
      {
        "status": "INITIALIZED",
        "descriptors": [],
        "oetype": "SOAP",
        "version": "11.4ALPHA",
        "description": "PAS OpenEdge SOAP Service.",
        "name": "SOAP",
        "state": "ENABLED",
        "type": "OPENEDGE",
        "uri": "http://localhost.com:16680/soap"
      }
    ]
  },
  "operation": "UNDEPLOY SOAP TRANSPORT DESCRIPTOR",
  "versionStr": "PASOE 11.5.0",
```

```
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg": ""
}
```

Command-line example (REST)

```
curl -X DELETE -v
http://localhost:16680/oemanager/applications/pas1/webapps/ROOT/transports/rest/oeservices/_oepingService
```

Response body example (REST)

```
{"result":
  {
    "OERestServiceDesc":[
      {
        "status":"INITIALIZED",
        "contextPath":"\/\/wrk\/pas1\/webapps\/ROOT",
        "descriptors":[
          {

"archiveLocation":"WEB-INF\/adapters\/rest\/_oepingService\/_oepingService.paar",
            "archiveName":"_oepingService.paar",
            "accessUrl":"\/rest\/_oepingService",
            "oetype":"REST_DESCRIPTOR",
            "version":"11.5.0",
            "description":"PAS OpenEdge REST Service Descriptor",
            "name":"_oepingService",
            "type":"OPENEDGE",
            "uri":"http:\/\/localhost.com:16680\/rest\/_oepingService"
          }
        ],
        "oetype":"REST",
        "version":"v11.4.0 (12-Dec-2013)",
        "description":"PAS OpenEdge REST Transport.",
        "name":"REST",
        "state":"DISABLED",
        "type":"OPENEDGE",
        "uri":"http:\/\/localhost.com:16680\/rest"
      }
    ]
  },
  "operation":"UNDEPLOY REST TRANSPORT DESCRIPTOR",
  "versionStr":"PASOE 11.5.0",
  "versionNo":1,
  "outcome":"SUCCESS",
  "errmsg":""
}
```

Application management

This section describes the REST services that are available for managing ABL and Web applications deployed on the Progress Application Server.

See also

List ABL application information on page 294
Get information about a specific ABL application on page 296
List Web applications deployed to an ABL application on page 298

Get information about a specific Web application on page 299

List ABL application information

Description

Get information about ABL applications from the OEManager.

HTTP Operation

```
GET
```

URI

```
//host_name:port/oemanager/applications/
```

Media type

```
application/vnd.progress+json
```

Request body

NA

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v http://localhost:16680/oemanager/applications/
```

Response body example

```
  "result": {
    "Application": [
      {
        "webapps": [
          {
            "applicationName": "pas1",
            "transports": [
              {
                "status": "INITIALIZED",
                "descriptors": [
                  {
                    "archiveLocation":
"/wrk/pas1/webapps/ROOT/WEB-INF/adapters/soap//ROOT/SimpleTest.wsdl",
                    "archiveName": "SimpleTest.wsm",
                    "accessUrl": "urn:nxgas:simpletestsf",
                    "oetype": "SOAP_DESCRIPTOR",
                    "version": "11.5.0",
                    "description": "PAS OpenEdge SOAP Service Artifact",
                    "name": "SimpleTest",
                    "state": "DISABLED",
                    "type": "OPENEDGE"
                  }
```

```
                  ],
                  "oetype": "SOAP",
                  "version": "11.4ALPHA",
                  "description": "PAS OpenEdge SOAP Service.",
                  "name": "SOAP",
                  "state": "DISABLED",
                  "type": "OPENEDGE",
                  "uri": "http://localhost:16680/soap"
                },
                {
                  "status": "INITIALIZED",
                  "oetype": "APSV",
                  "version": "v115.PASOE (11-SEPT-2014)",
                  "description": "PAS OpenEdge APSV Transport.",
                  "name": "APSV",
                  "state": "ENABLED",
                  "type": "OPENEDGE",
                  "uri": "http://localhost:16680/apsv"
                },
                {
                  "status": "INITIALIZED",
                  "contextPath": "//wrk/pas1/webapps/ROOT",
                  "descriptors": [
                    {
                      "archiveLocation":
"WEB-INF/adapters/rest/_oepingService/_oepingService.paar",
                      "archiveName": "_oepingService.paar",
                      "accessUrl": "/rest/_oepingService",
                      "oetype": "REST_DESCRIPTOR",
                      "version": "11.5.0",
                      "description": "PAS OpenEdge REST Service Descriptor",
                      "name": "_oepingService",
                      "type": "OPENEDGE",
                      "uri": "http://localhost:16680/rest/_oepingService"
                    }
                  ],
                  "oetype": "REST",
                  "version": "v11.4.0 (12-Dec-2013)",
                  "description": "PAS OpenEdge REST Transport.",
                  "name": "REST",
                  "state": "ENABLED",
                  "type": "OPENEDGE",
                  "uri": "http://localhost:16680/rest"
                }
              ],
              "oetype": "WEBAPP",
              "version": "11.5.0",
              "description": "PAS OpenEdge WebApp",
              "name": "ROOT",
              "state": "STARTED",
              "type": "OPENEDGE",
              "uri": "http://localhost:16680/"
            }
          ],
          "oetype": "APPLICATION",
          "version": "11.5.0",
          "description": "PAS Application",
          "name": "pas1",
          "type": "OPENEDGE"
        }
      ]
    },
    "operation": "GET OEABL SERVICES",
    "versionStr": "PASOE 11.5.0",
    "versionNo": 1,
    "outcome": "SUCCESS",
    "errmsg": ""
  }
```

Get information about a specific ABL application

Description

Get information about a specified application from the OEManager.

HTTP Operation

GET

URI

//host_name:*port*/oemanager/applications/*ABL_app_name*

Media type

application/vnd.progress+json

Request body example

NA

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v http://localhost:16680/oemanager/applications/pas1
```

Response body example

```
{
  "result": {
    "Application": [
      {
        "webapps": [
          {
            "applicationName": "pas1",
            "transports": [
              {
                "status": "INITIALIZED",
                "descriptors": [
                  {
                    "archiveLocation":
"/wrk/pas1/webapps/ROOT/WEB-INF/adapters/soap//ROOT/SimpleTest.wsdl",
                    "archiveName": "SimpleTest.wsm",
                    "accessUrl": "urn:nxgas:simpletestsf",
                    "oetype": "SOAP_DESCRIPTOR",
                    "version": "11.5.0",
                    "description": "PAS OpenEdge SOAP Service Artifact",
                    "name": "SimpleTest",
                    "state": "DISABLED",
                    "type": "OPENEDGE"
                  }
                ],
                "oetype": "SOAP",
```

```
                  "version": "11.4ALPHA",
                  "description": "PAS OpenEdge SOAP Service.",
                  "name": "SOAP",
                  "state": "DISABLED",
                  "type": "OPENEDGE",
                  "uri": "http://localhost:16680/soap"
                },
                {
                  "status": "INITIALIZED",
                  "oetype": "APSV",
                  "version": "v115.PASOE (11-SEPT-2014)",
                  "description": "PAS OpenEdge APSV Transport.",
                  "name": "APSV",
                  "state": "ENABLED",
                  "type": "OPENEDGE",
                  "uri": "http://localhost:16680/apsv"
                },
                {
                  "status": "INITIALIZED",
                  "contextPath": "/wrk/pas1/webapps/ROOT",
                  "descriptors": [
                    {
                      "archiveLocation":
"WEB-INF/adapters/rest/_oepingService/_oepingService.paar",
                      "archiveName": "_oepingService.paar",
                      "accessUrl": "/rest/_oepingService",
                      "oetype": "REST_DESCRIPTOR",
                      "version": "11.5.0",
                      "description": "PAS OpenEdge REST Service Descriptor",
                      "name": "_oepingService",
                      "type": "OPENEDGE",
                      "uri": "http://localhost:16680/rest/_oepingService"
                    }
                  ],
                  "oetype": "REST",
                  "version": "v11.4.0 (12-Dec-2013)",
                  "description": "PAS OpenEdge REST Transport.",
                  "name": "REST",
                  "state": "ENABLED",
                  "type": "OPENEDGE",
                  "uri": "http://localhost:16680/rest"
                }
              ],
              "oetype": "WEBAPP",
              "version": "11.5.0",
              "description": "PAS OpenEdge WebApp",
              "name": "ROOT",
              "state": "STARTED",
              "type": "OPENEDGE",
              "uri": "http://localhost:16680/"
            }
          ],
          "oetype": "APPLICATION",
          "version": "11.5.0",
          "description": "PAS Application",
          "name": "pas1",
          "type": "OPENEDGE"
        }
      ]
    },
    "operation": "GET OEABL SERVICE",
    "versionStr": "PASOE 11.5.0",
    "versionNo": 1,
    "outcome": "SUCCESS",
    "errmsg": ""
  }
```

List Web applications deployed to an ABL application

Description

Get information from the OEManager about all the Web applications deployed to an ABL application.

HTTP Operation

GET

URI

//host_name:*port*/oemanager/applications/*ABL_app_name*/webapps

Media type

application/vnd.progress+json

Request body

NA

Response codes

200 Success
500 Unexpected Server Error

Command-line example

```
curl -X GET -v http://localhost:16680/oemanager/applications/pas1/webapps
```

Response body example

```
{
  "result": {
    "WebApp": [
      {
        "applicationName": "pas1",
        "transports": [
          {
            "status": "INITIALIZED",
            "descriptors": [
              {
                "archiveLocation":
"/wrk/pas1/webapps/ROOT/WEB-INF/adapters/soap//ROOT/SimpleTest.wsdl",
                "archiveName": "SimpleTest.wsm",
                "accessUrl": "urn:nxgas:simpletestsf",
                "oetype": "SOAP_DESCRIPTOR",
                "version": "11.5.0",
                "description": "PAS OpenEdge SOAP Service Artifact",
                "name": "SimpleTest",
                "state": "DISABLED",
                "type": "OPENEDGE"
              }
            ],
            "oetype": "SOAP",
            "version": "11.4ALPHA",
            "description": "PAS OpenEdge SOAP Service.",
```

```
              "name": "SOAP",
              "state": "DISABLED",
              "type": "OPENEDGE",
              "uri": "http://localhost:16680/soap"
            },
            {
              "status": "INITIALIZED",
              "oetype": "APSV",
              "version": "v115.PASOE (11-SEPT-2014)",
              "description": "PAS OpenEdge APSV Transport.",
              "name": "APSV",
              "state": "ENABLED",
              "type": "OPENEDGE",
              "uri": "http://localhost:16680/apsv"
            },
            {
              "status": "INITIALIZED",
              "contextPath": "/wrk/pas1/webapps/ROOT",
              "descriptors": [
                {
                  "archiveLocation":
"WEB-INF/adapters/rest/_oepingService/_oepingService.paar",
                  "archiveName": "_oepingService.paar",
                  "accessUrl": "/rest/_oepingService",
                  "oetype": "REST_DESCRIPTOR",
                  "version": "11.5.0",
                  "description": "PAS OpenEdge REST Service Descriptor",
                  "name": "_oepingService",
                  "type": "OPENEDGE",
                  "uri": "localhost:16680/rest/_oepingService"
                }
              ],
              "oetype": "REST",
              "version": "v11.4.0 (12-Dec-2013)",
              "description": "PAS OpenEdge REST Transport.",
              "name": "REST",
              "state": "ENABLED",
              "type": "OPENEDGE",
              "uri": "localhost:16680/rest"
            }
          ],
          "oetype": "WEBAPP",
          "version": "11.5.0",
          "description": "PAS OpenEdge WebApp",
          "name": "ROOT",
          "state": "STARTED",
          "type": "OPENEDGE",
          "uri": "localhost:16680/"
        }
      ]
    },
    "operation": "GET OEABL SERVICES",
    "versionStr": "PASOE 11.5.0",
    "versionNo": 1,
    "outcome": "SUCCESS",
    "errmsg": ""
}
```


Get information about a specific Web application

Description

Get information from the OEManager about a specific Web application deployed to an ABL application.

HTTP Operation

GET

URI

//host_name:*port*/oemanager/applications/*ABL_app_name*/webapps/*Web_app_name*

Media type

application/vnd.progress+json

Request body

NA

Response codes

200 Success
500 Unexpected Server Error

Command-line example

```
curl -X GET -v http://localhost:16680/oemanager/applications/pas1/webapps/ROOT
```

Response body example


```
{
  "result": {
    "WebApp": [
      {
        "applicationName": "pas1",
        "transports": [
          {
            "status": "INITIALIZED",
            "descriptors": [
              {
                "archiveLocation":
"/wrk/pas1/webapps/ROOT/WEB-INF/adapters/soap//ROOT/SimpleTest.wsdl",
                "archiveName": "SimpleTest.wsm",
                "accessUrl": "urn:nxgas:simpletestsf",
                "oetype": "SOAP_DESCRIPTOR",
                "version": "11.5.0",
                "description": "PAS OpenEdge SOAP Service Artifact",
                "name": "SimpleTest",
                "state": "DISABLED",
                "type": "OPENEDGE"
              }
            ],
            "oetype": "SOAP",
            "version": "11.4ALPHA",
            "description": "PAS OpenEdge SOAP Service.",
            "name": "SOAP",
            "state": "DISABLED",
            "type": "OPENEDGE",
            "uri": "http://localhost:16680/soap"
          },
          {
            "status": "INITIALIZED",
            "oetype": "APSV",
            "version": "v115.PASOE (11-SEPT-2014)",
            "description": "PAS OpenEdge APSV Transport.",
```

```
            "name": "APSV",
            "state": "ENABLED",
            "type": "OPENEDGE",
            "uri": "http://localhost:16680/apsv"
          },
          {
            "status": "INITIALIZED",
            "contextPath": "/wrk/pas1/webapps/ROOT",
            "descriptors": [
              {
                "archiveLocation":
"WEB-INF/adapters/rest/_oepingService/_oepingService.paar",
                "archiveName": "_oepingService.paar",
                "accessUrl": "/rest/_oepingService",
                "oetype": "REST_DESCRIPTOR",
                "version": "11.5.0",
                "description": "PAS OpenEdge REST Service Descriptor",
                "name": "_oepingService",
                "type": "OPENEDGE",
                "uri": "http://localhost:16680/rest/_oepingService"
              }
            ],
            "oetype": "REST",
            "version": "v11.4.0 (12-Dec-2013)",
            "description": "PAS OpenEdge REST Transport.",
            "name": "REST",
            "state": "ENABLED",
            "type": "OPENEDGE",
            "uri": "http://localhost:16680/rest"
          }
        ],
        "oetype": "WEBAPP",
        "version": "11.5.0",
        "description": "PAS OpenEdge WebApp",
        "name": "ROOT",
        "state": "STARTED",
        "type": "OPENEDGE",
        "uri": "http://localhost:16680/"
      }
    ]
  },
  "operation": "GET OEABL SERVICES",
  "versionStr": "PASOE 11.5.0",
  "versionNo": 1,
  "outcome": "SUCCESS",
  "errmsg": ""
}
```


Monitoring memory use

This section describes the REST services that are available for monitoring memory use of a multi-session agent.

Track memory use

Description

Toggle tracking of memory use of a multi-session agent, on or off.

HTTP Operation

POST

URI

//host_name:*port*/oemanager/applications/*App_name*/agents/*agentID/*trackMemoryUse

/*true-or-false*

Media type

application/vnd.progress+json

Response codes

200 Success
403 Access Denied
500 Unexpected Server Error

Command-line example

```
curl -X POST -v
http://localhost:16680/oemanager/applications/oepas1/agents/CPnXPDNcSNGOCzVHqz9o6Q/trackMemoryUse/true
```

Request body example

NA

Response body example

NA

Tracking memory use

Description

Check if memory use of a multi-session agent is being currently tracked.

HTTP Operation

GET

URI

//host_name:*port*/oemanager/applications/*App_name*/agents/*agentID/*trackingMemoryUse

Media type

application/vnd.progress+json

Response codes

```
200 Success
403 Access Denied
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v
http://localhost:16680/oemanager/applications/oepas1/agents/CPnXPDNcSNGOCzVHqz9o6Q/trackingMemoryUse
```

Request body example

NA

Response body example

```
{
    "result": true,
    "operation": "TRACKING MEMORY USE",
    "versionStr": "v11.7.3 ( 2018-02-12 )",
    "errmsg": "",
    "versionNo": 1,
    "outcome": "SUCCESS"
}
```

Generate memory use report for an agent

Description

Generate a report of memory use of all the ABL sessions of a multi-session agent.

HTTP Operation

GET

URI

//host_name:*port*/oemanager/applications/*App_name*/agents/*agentID/*getMemoryUseReport

Media type

application/vnd.progress+json

Response codes

```
200 Success
403 Access Denied
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v
http://localhost:16680/oemanager/applications/oepas1/agents/CPnXPDNcSNGOCzVHqz9o6Q/getMemoryUseReport
```

Request body example

NA

Response body example


```
{
    "result": {
        "ABLOutput": {
            "Memuse_info": {
                "DateTime": "02/27/18 16:44:20.196+00:00   ",
                "Overhead": [
                    {
                        "Src": "csnet.c",
                        "Line": 4536,
                        "Amt": 1052,
                        "AllocCnt": 1
                    },
                    {
                        "Src": "csmtadmin.c",
                        "Line": 485,
                        "Amt": 40,
                        "AllocCnt": 1
                    },
                    {
                        "Src": "csmtpools.c",
                        "Line": 165,
                        "Amt": 64,
                        "AllocCnt": 1
                    }
                ],
                "Sessions": [
                                        {
                        "AgentSessionId": 1,
                        "SessionLogId": "AS-Listener",
                        "SessionMemory": 525817,
                        "MaxPoolTrackLen": 0,
                        "FreedOrClearedPools": 1,
                        "AvgPoolTrackLen": "0.00",
                        "Memuse_data": []
                    }
                ],
                "UniqueAllocationPoints": 236,
                "AvgSrcLineBktDepth": "1.12",
                "MaxSrcLineBktDepth": 2,
                "MaxPoolTrackLen": 10,
                "FreedOrClearedPools": 52,
                "AvgPoolTrackLen": "2.12"
            }
        },
        "ABLReturnVal": true,
        "agentId": "CPnXPDNcSNGOCzVHqz9o6Q",
        "pid": "19644"
    },
    "operation": "GET MEMORY USE REPORT",
    "versionStr": "v11.7.3 ( 2018-02-12 )",
    "errmsg": "",
    "versionNo": 1,
    "outcome": "SUCCESS"
}
```

Generate memory use report for an ABL session

Description

Generate a report of memory use of a specific ABL session of a multi-session agent.

HTTP Operation

GET

URI

//*host_name*:*port*/oemanager/applications/*App_name*/agents/*agentID*/getMemoryUseReport/sessions/*agentsessionID*

Media type

application/vnd.progress+json

Response codes

```
200 Success
403 Access Denied
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -v
http://localhost:16680/oemanager/applications/oepas1/agents/CPnXPDNcSNGOCzVHqz9o6Q/getMemoryUseReport/sessions/1
```

Request body example

NA

Response body example


```
{
    "result": {
        "ABLOutput": {
            "Memuse_info": {
                "DateTime": "02/27/18 17:04:30.660+00:00   ",
                "Overhead": [
                    {
                        "Src": "csnet.c",
                        "Line": 4536,
                        "Amt": 1052,
                        "AllocCnt": 1
                    },
                    {
                        "Src": "csmtadmin.c",
                        "Line": 485,
                        "Amt": 40,
                        "AllocCnt": 1
                    },
                    {
                        "Src": "csmtmemuse.c",
                        "Line": 819,
                        "Amt": 8,
                        "AllocCnt": 1
```

```
                    }
                ],
                "Sessions": [
                    {
                        "AgentSessionId": 1,
                        "SessionLogId": "AS-Listener",
                        "SessionMemory": 525817,
                        "MaxPoolTrackLen": 0,
                        "FreedOrClearedPools": 1,
                        "AvgPoolTrackLen": "0.00",
                        "Memuse_data": []
                    }
                ],
                "UniqueAllocationPoints": 236,
                "AvgSrcLineBktDepth": "1.12",
                "MaxSrcLineBktDepth": 2,
                "MaxPoolTrackLen": 10,
                "FreedOrClearedPools": 53,
                "AvgPoolTrackLen": "2.17"
            }
        },
        "ABLReturnVal": true,
        "agentId": "CPnXPDNcSNGOCzVHqz9o6Q",
        "pid": "19644"
    },
    "operation": "GET MEMORY USE REPORT",
    "versionStr": "v11.7.3 ( 2018-02-12 )",
    "errmsg": "",
    "versionNo": 1,
    "outcome": "SUCCESS"
}
```

Index

A

ABL application information *294*, *296*, *298–299*
ABL applications *92*
Add additional index entries here (if needed)
    Nest index terms for sub entries *53*
Administrative tools *35*
Agent management *256*
Agent status *260*
Amazon load balancing *109*
Apache load balancing *109*
Apache proxy host load balancing *101*
Apache proxy load balancing *102–103*
Application management *293*
Audience *11*
Auto scaling *109*

C

Cancel a request *252*
clean action *224*
Clients *245*
Clusters *99*
config action *220*
Configuration files *24*
Configuration properties *248–249*, *267–268*
Connection information *262*
CORS support *139*
create *217*
Creating instances *84*

D

delete *219*
deploy *207*
Deployed services *282*
deployREST.sh *90*
deploySOAP.sh *91*
Development server *112*
Development servers *20*
Directory structure extensions *33*
disable *214*
Dump a session stack *253*

E

Elastic load balancing *109*
enable *213*
Enabling Spring Security *126*
env *240*

F

feature action *222*

H

headertest *156*
help *240*
HTTP sessions *97*
HTTPS *112–113*

I

info *206*
instances *224*
Instances
    creating *84*
    managing *83*
    Running as a UNIX daemon *86*
    Running as a Windows service *86*

J

JConsole *56*
JMX *56*
JVM security manager *119*

L

LDAP security configurations *128*
leaks *212*
list *205*
List agents *256*
List clients *245*
List sessions *244*
Load balancing
    Apache proxy host *101*
    Apache proxy load balancing *102*
    Tomcat load balancing *105*
Logging *23*
login models *138*

M

Managing instances *83*
Managing Web applications *89*
MBeans *56*
MemoryRealm *120*

O

OEJMX *71*
OEPROP *50*
OERealm *129*
OpenEdge Explorer *56*
OpenEdge Management *56*
Organization *12*

P

PAS for OpenEdge
  development server *112*
  production server *112*
PASMAN *43*
PASOE
  administrative utilities *35*
pasoestart *37*
plist *226*
Production server *112*
Production servers *20*
Progress Application Server
  directory structure extensions *33*
  extensions to Apache Tomcat *19*
Progress Application Server for OpenEdge
  load balancing *101*
Properties
  PAS properties *25*
Properties files *24*
Proxy load balancing *104*
Purpose *11*

R

Realms *119*
register *227*
reload *210*
Remote access filters *121*
Request information *247*, *264*
Required: add at least one index entry *53*
Reset runtime metrics *251*
resources *215*
REST API for oemanager.war
  cancel a request *252*
  clients *245*
  deploy a new service *286–287*
  dump a session stack *253*
  enable or disable a service *291*
  enable or disable a transport *275–276*
  get ABL application information *294*, *296*, *299*
  get agent status *260*
  get configuration properties *248*, *267*
  get connection information *262*
  get request information *247*, *264*
  get runtime metrics *250*
  get service configuration properties *285*
REST API for oemanager.war *(continued)*
  get service information *283*
  get service runtime metrics *289*
  get session metrics *261*
  get transport configuration properties *272*
  get transport metrics *277*, *279*
  list agents *256*
  list deployed services *282*
  list Web applications *298*
  removing a service *292*
  reset runtime metrics *251*
  reset service runtime metrics *290*
  reset transport metrics *280–281*
  sessions *244*
  stop an agent *270*
  terminate a session *254*
  terminate idle ABL sessions *255*
  update configuration properties *249*, *268*
  update service runtime properties *288*
  update transport properties *274*
REST API Reference *243*
REST service deployment (if needed)
  Nest index terms for sub entries *90*
Roles
  assigning to users *120*
ROOT Web application *20*
Runtime metrics *250*

S

Security *111*
service *228*
Service deployment *286–287*
Service enable or disable *291*
Service information *283*, *285*
Service removal *292*
Service runtime metrics *289–290*
Service runtime properties *288*
Session management *244*
Session metrics *261*
sessions *216*
Sessions *244*
showproc *230*
SOAP service deployment *91*
Spring Security *119*
SSO support *140*
start *231*
status *211*
Sticky sessions *99*
stop *233*
Stop an agent *270*
stsconnroleutil
  disable *159*
  enable *158*
  grantfile *161*
  grantuser *160*

stsconnroleutil *(continued)*
list *159*
revokefile *162*
revokeuser *163*
status *157*
stskeyutil
create *152*
keyfile *154*
ssotest *156*
stskeyutil verify *153*
stskeyutilkeytest *155*
stsurlutil
list *164*
status *164*
update *165*

T

tcman *200*
TCMAN
clean *224*
config *220*
create *217*
customizing *202*
delete *219*
deploy *207*
disable *214*
enable *213*
env *240*
extending *202*
feature *222*
general actions *239*
help *240*
info *206*
instances *224*
leaks *212*
list *205*, *226*
manager actions *204*
register *227*
reload *210*
resources *215*
server actions *216*
service *228*
sessions *216*
showproc *230*
start *231*
status *211*
TCMAN *(continued)*
stop *233*
test *235*
undeploy *209*
unregister *236*
using *199*
version *234*
worker *237*
Terminate a session *254*
Terminate idle ABL sessions *255*
test *235*
Tomcat load balancing *105–107*
Transport management
URLs *20*
Transport metrics *277*, *279–281*
Transport properties *272*, *274*
Transports *275–276*

U

undeploy *209*
UNIX daemon *86*
unregister *236*
URLs
WEB transport *21*

V

version *234*

W

Web application deployment *90*
Web application security configurations *123*
Web applications
managing *89*
Packaging *91*
WEB transport URL *21*
WebSpeed
configuration *193–194*
configuring openedge.properties *195*
management *193–194*
managing application files *197*
security *196*
Windows service *86*
worker *237*