Progress®



Manage Progress Application Server (PAS) for OpenEdge

OpenEdge®

Copyright

© 2021 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.

DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Defrag This, Deliver More Than Expected, DevReach (and design), Icenium, Inspec, Ipswitch, iMacros, Kendo UI, Kinvey, MessageWay, MOVEit, NativeChat, NativeScript, OpenEdge, Powered by Chef, Powered by Progress, Progress, Progress Software Developers Network, SequeLink, Sitefinity (and Design), Sitefinity, Sitefinity (and design), SpeedScript, Stylus Studio, Stylized Design (Arrow/3D Box logo), Styleized Design (C Chef logo), Stylized Design of Samurai, TeamPulse, Telerik, Telerik (and design), Test Studio, WebSpeed, WhatsConfigured, WhatsConnected, WhatsUp, and WS_FTP 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, Chef Automate, Chef Compliance, Chef Desktop, Chef Habitat, Chef WorkStation, Corticon.js, Corticon Rules, Data Access, DataDirect Autonomous REST Connector, DataDirect Spy, DevCraft, Fiddler, Fiddler Everywhere, FiddlerCap, FiddlerCore, FiddlerScript, Hybrid Data Pipeline, iMail, JustAssembly, JustDecompile, JustMock, KendoReact, NativeScript Sidekick, OpenAccess, PASOE, Pro2, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Push Jobs, SafeSpaceVR, Sitefinity Cloud, Sitefinity CMS, Sitefinity Digital Experience Cloud, Sitefinity Feather, Sitefinity Insight, Sitefinity Thunder, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Supermarket, SupportLink, Unite UX, 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.

July 2021

Product version: Progress OpenEdge 12.4

Updated: 2021/07/09

Table of Contents

Preface....11

Learn about PAS for OpenEdge administration....13

PAS for OpenEdge product modes....14
About development and production instances....14
Missing and disabled features in production....15
About security models....16
About licensing and installation....17
About the PAS for OpenEdge configuration tools....18
Tomcat utilities....18
TCMAN....18
PASOESTART....19
PASMAN....25
SECPROP....28
OEPROP....34
PASPROPCONV....36
Utilities to manage keys and digital certificates....38
OpenEdge Management and OpenEdge Explorer....39
JConsole and JMX....39
OEJMX utility....43
Before you move to production....45
Production checklist....46

Create and configure PAS for OpenEdge instances....47

Create a PAS for OpenEdge production instance....48
Use PASMAN to create an instance....48
Use OpenEdge Management to create an instance....49
Check the security model of the instance....50
Start and stop a PAS for OpenEdge instance....50
Use PASMAN to start and stop an instance....51
Use OpenEdge Management to start and stop an instance....51
Run PAS for OpenEdge as a non-root user....51
Automate lifecycle event scripts....54
Configure a PAS for OpenEdge instance....54
Use PASMAN to configure an instance....55
Use OpenEdge Management to configure an instance....56
Connect to an OpenEdge database from PAS for OpenEdge....56
Register a PAS for OpenEdge instance as a Windows service....58
Configure private PAS for OpenEdge instance tracking....59

Delete a PAS for OpenEdge instance....62
Use PASMAN to delete an instance....62
Use OpenEdge Management to delete an instance....62

Deploy an application in PAS for OpenEdge....63

Configure properties for deployment....64
Set the PROPATH....64
Set the database connections....64
Enable the transports....65
Set the agents and sessions....65
Modify the environment variables....66
Stage the r-code....66
OpenEdge Application Archive Structure....67
Create an OpenEdge Application Archive using tcman export....69
Create an OpenEdge Application Archive using an Ant Build....70
Tailor an ABLApp installation using Ant....73
Deployment artifacts....74
Deploy an OpenEdge Application Archive using tcman import....75
Deploy REST services....77
Deploy ABL services....77
Deploy SOAP services....77
Deploy web handler services....78
Deploy an ABL web application 81
Deploy remote administration applications....82
ABL Service Deployment....82
ABL application PING service....85
Package an instance for production....88

Secure PAS for OpenEdge instances....89

About customizing security for a production instance....91
Enable ABL application authentication....92
About the URL access control file....96
About local and extended local user accounts....96
About LDAP user accounts....98
About Active Directory user accounts....98
About OERealm user account configurations....99
About multi-domain support for user accounts....120
Disable HTTP in production instances....124
Use TLS in PAS for OpenEdge....124
Obtain a certificate from a certificate authority....125
Configure clients for TLS....127
Configure a PAS for OpenEdge instance for TLS....128
About client authentication....130

Configure ABL client authentication....131
Configure a PAS for OpenEdge instance for client authentication....132
Configure Spring Security....135
About single sign-on support....136
About CORS support....136
Add CORS support....137
About managing users and remote access....137
About user and file permissions....138
About role-based user authentication....138
About remote access filters....140
Authenticate using OpenEdge Authentication Gateway as a security token service....140
Authenticate using OAuth2 and JWT....140
About OAuth2....141
About OAuth2 security considerations....143
About OAuth2 tokens....143
About JSON Web Tokens....145
About support for OAuth2 and JWT in PAS for OpenEdge....148
Configure a PAS for OpenEdge web application as an OAuth2 Resource Server....150
Debug logging for OAuth2....158
Authenticate using SAML....158
About SAML....159
About support for SAML in PAS for OpenEdge....161
Implement SAML in a PAS for OpenEdge web application....163
Debug logging for SAML....166
Extend OpenEdge SSO to web applications....166
About PAS for OpenEdge SSO technologies....166
PAS for OpenEdge SSO configuration guide....168
PAS for OpenEdge guide to SSO token handling....181
About support for server name indication....185
STS client utility (stsclientutil)....185
stsurlutil update utility....188

Tune PAS for OpenEdge instances....191

PAS for OpenEdge tuning recommendations....192
Goals and common steps for tuning PAS for OpenEdge instances....194
PAS for OpenEdge server-side tuning....195
JVM parameters....195
Network connections....197
HTTP/HTTPS request processing....198
PAS for OpenEdge instance parameters....199
Tune OpenEdge web applications and multi-session agents....201
ABL session manager and session pool....201
Web application startup parameters....204

Load balance PAS for OpenEdge instances.....207

Scaling considerations in PAS for OpenEdge.....209
Enable HTTP sessions for the APSV transport.....209
Clusters and sticky sessions.....210
Apache proxy host load balancing.....212
Requirements for Apache proxy load balancing.....212
Configure Apache for proxy load balancing.....213
Configure PAS for OpenEdge instances for proxy load balancing.....214
Tomcat load balancing.....215
Requirements for Tomcat load balancing.....216
Configure an Apache web server for Tomcat load balancing.....216
Configure PAS for OpenEdge for Tomcat load balancing.....217
Amazon load balancing.....219
Configure Elastic Load Balancing and Auto Scaling.....219
Docker container load balancing.....221

Monitor PAS for OpenEdge instances.....223

Configure logging in PAS for OpenEdge.....224
Configure Tomcat and Tomcat web application logging.....225
Configure PAS for OpenEdge application logging.....227
Configure ABL application logging.....230
Configure ABL application agent logging.....231
Configure log file patterns.....234
Configure MDC access for header requests.....240
Configure the log file directory path.....241
Configure log rollover settings.....242
Use deferred logging in PAS for OpenEdge.....242
Use OEJMX to manage and monitor an instance.....244
List available MBeans to locate AgentManager details.....244
Use an OEJMX query to get agent information.....246
Use an OEJMX query to get session metrics.....248
Common OEJMX queries.....250
Use Server-Side ABL Performance Profiling.....251
What is Server-Side Profiling?.....251
Configure the diagnostic store.....252
Configure Server-Side Profiling with PAS for OpenEdge.....253
Supported JSON values for JMX queries.....256
Use Swagger UI to explore management REST APIs 258
Enable Swagger UI for management REST API access.....258
Explore and test management REST APIs.....259
About the OpenEdge HealthScanner.....259
Use the OpenEdge HealthScanner.....261

Access HealthScanner data....264
Display a healthy score for a REST call....268
Display an unhealthy score for a REST call....271
Customize HealthScanner reports....274
Write a custom HealthScanner view....275
Collect runtime metrics....279
Enable debugging....289
About session and request states....290
Troubleshoot problems with an instance....291
Graceful emergency shutdown of a multi-session agent....292
Find memory leaks using ABL object tracking....293

Optimize PAS for OpenEdge for continuous operations....295

Secure online deployment of a new ABL application 296
Update PROPATH in a production instance with zero downtime....297
Refresh agents in an ABL application....298
Enhanced ABL session management....298

Index....303

Preface

Purpose

This content explains managing Progress Application Server for OpenEdge (PAS for OpenEdge) and focuses on the use of command-line utilities (TCMAN, PASMAN) and management APIs (oemanager, oejmx). For information about administration with OpenEdge Management or OpenEdge Explorer, see *Manage Progress Application Server for OpenEdge with OpenEdge Management*.

Audience

This content is intended for any OpenEdge application server administrators or ABL developers who need to create, configure, or manage instances of PAS for OpenEdge.

Organization

This content is organized in the following way:

- Learn about PAS for OpenEdge administration on page 13

  Introduces the administrative tools and concepts for PAS for OpenEdge. PAS for OpenEdge administration tools are also described in *Progress Application Server for OpenEdge Administrative Tools Reference*.

- Create and configure a PAS for OpenEdge instance

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

- Deploy an application in PAS for OpenEdge on page 63

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

- Secure a PAS for OpenEdge instance

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

- Tune PAS for OpenEdge instances

  Describes PAS for OpenEdge instance tuning techniques and recommendations.

- Load balance PAS for OpenEdge instances on page 207

  Describes configuring load balancing options for PAS for OpenEdge instances.

- Monitor PAS for OpenEdge instances

  Describes the monitoring tools available in PAS for OpenEdge.

Documentation conventions

See Documentation Conventions for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content library.

1

Learn about PAS for OpenEdge administration

The following topics introduce you to the overall administrative framework of Progress Application Server for OpenEdge (PAS for OpenEdge), including:

- Installing and configuring PAS for OpenEdge.
- Moving a PAS for OpenEdge instance to production.
- Application deployment on a PAS for OpenEdge instance.
- Securing a PAS for OpenEdge instance.
- Tuning, load balancing, and monitoring a PAS for OpenEdge instance.

For details, see the following topics:

- PAS for OpenEdge product modes
- About development and production instances
- Missing and disabled features in production
- About security models
- About licensing and installation
- About the PAS for OpenEdge configuration tools
- Before you move to production
- Production checklist

PAS for OpenEdge product modes

PAS for OpenEdge can be installed as either a server for developing and testing web applications or as a production server for application deployment. PAS for OpenEdge has three products with different licensing:

- A development mode product: Progress Development Application Server for OpenEdge (`Progress Dev AS for OE` in the licensing) is configured as a web server for developing and testing OpenEdge applications.
- A production mode product: Progress Production Application Server for OpenEdge (`Progress Prod AS for OE` in the licensing) is configured as a secure web server for OpenEdge application deployment.
- A limited production mode product: Progress Application Server for OpenEdge Lite (`PASOE Lite` in the licensing) is configured as a secure web server for OpenEdge application deployment.

  For more information, see "PAS for OpenEdge Lite" in *Learn about Progress Application Server (PAS) for OpenEdge*.

The development mode product comes with the base OpenEdge installation, while the production mode product requires an additional license.

A production instance of PAS for OpenEdge is not meant to be used out of the box. The production product enforces the current security best practices followed by many other current web-based products. You must make a number of configuration changes before you can run and deploy your applications to a production instance.

Progress recommends that you stage your instance using a production license. The staging environment is a test sandbox that replicates the production servers and environment.

About development and production instances

The difference between a development instance and a production instance of PAS for OpenEdge is mainly a matter of security configuration.

- An instance in a development environment allows unrestricted access to and control by a user or a group of users.
- An instance in a production environment is configured to restrict access to everyone except authorized users and to limit control to system administrators.

For example, in a development environment, anyone may be allowed to shut down the server, whereas, in a production environment, only a system administrator with the appropriate privileges is allowed to stop a running server. Also, be aware that the development server includes a number of Tomcat and Progress configuration and management tools. A production server, to avoid unauthorized tampering, does not include those tools by default.

A default PAS for OpenEdge development instance contains the following:

- An ABL web application deployed as `ROOT`. This application is derived from the `oeabl.war` application and contains a default ABL application that is name of the instance in the `ablapps` directory.
- The Tomcat Manager web application deployed as `manager`. Tomcat Manager provides monitoring and administration tools. Tomcat Manager should only be deployed in the development mode.
- The OpenEdge Manager web application is deployed as `oemanager`. OpenEdge Manager provides a REST API for remote management and monitoring of ABL applications deployed on a PAS for OpenEdge instance.
- The Progress Application Server Debugger for OpenEdge web application deployed as `oedbg`. This application enables debugging support for ABL applications deployed in PAS for OpenEdge development environments.

For a production security model instance, there is only an `oeabl.war`-based ROOT ABL web application that has all transports disabled by default. In OpenEdge 12.2, PAS for OpenEdge offers a subset of the production security instance that uses a `noaccess.war`-based ROOT ABL web application that allows for no access to any other web application. For more information, see Security models.

The differences between PAS for OpenEdge development and production instances are summarized in the following table:

PAS for OpenEdge development instance	PAS for OpenEdge production instance
Can compile code*	Cannot compile code*
Restricted to 5 concurrent requests and 1 agent*	Unrestricted concurrent requests and agents*
Non-secure configuration	Locked-down configuration
Test server instance in `$WRKDDIR`	No test server instances
Remote administration included with Tomcat Manager and OpenEdge Manager applications	Remote administration optional with Tomcat Manager and OpenEdge Manager applications
Built-in `oeabl` web application (ROOT) with all transports deployed and enabled	Built-in `oeabl` web application (ROOT) with all transports deployed but disabled

Note: The differences indicated with (*) are enforced by the license agreements.

Missing and disabled features in production

Because the production version of PAS for OpenEdge should be as secure as possible after the initial installation, all debugging and management features are disabled by default. They can be re-enabled if needed.

The following is a list of the disabled features in the production mode of PAS for OpenEdge:

- **Status home page is not implemented.**

  The development mode of PAS for OpenEdge includes a default home page that displays status information. The home page is provided for debugging purposes so the developer has quick access to server data. In order to prevent internal data from being viewed externally, there is no default status page in the production mode.

- **Status responses are off by default.**

  Status response is controlled by the `statusEnabled` property in the `openedge.properties` file. If enabled, status is returned in a JSON page when you query the transport with the following syntax:

  `http://`*`hostname`*`:`*`port`*`/apsv|rest|soap|web`

- **Tomcat Manager and OpenEdge REST API Manager are not deployed.**

  If you want to deploy these remote manager applications, see Deploy remote administration applications.

About security models

Development and production instances are specified at instance creation time, and cannot be changed from one security model to another. Prior to OpenEdge 12.2, PAS for OpenEdge supported two security models: **dev** and **prod**.

The security model can be specified during instance creation with the `-Z` switch. See the following examples:

```
proenv>pasman create -Z dev -f -p 8820 -P 8821 myDevInst
proenv>pasman create -Z prod -p 8830 -P 8831 myProdInst
```

Note: The creation of `myDevInst` also uses the `-f` switch, which deploys monitoring applications such as Tomcat Manager (`manager.war`) and OpenEdge Manager (`oemanager.war`) to the instance. These monitoring applications are useful in development, but should never be deployed to a production instance due to security risks.

Differences between these two security models and their licensing are described in Development versus production instances and Licensing and installation.

OpenEdge 12.2 introduces a third security model, which is a subset of the production security model: **pas**. To create a PAS for OpenEdge instance with the pas production security model, see the following example:

```
proenv>pasman create -Z pas -p 8840 -P 8841 mySecuredInst
```

The pas production security model creates an instance with `noaccess.war` deployed to ROOT, rather than the standard `oeabl.war`. The `noaccess.war` file provides the ability to reserve a PAS for OpenEdge server URL, and allow no access to any other web application, ABL web application, or ABL business application. You can start the instance, but no ABL agent process is started because no ABL application is deployed to this instance, and by default the browser returns a 400 “request refused” error when called. More information can be found in the `noaccess.README` file in the *`instance-name`*`/webapps/ROOT/WEB-INF` directory of an instance deployed with the pas production security model.

The primary use case for `noaccess.war` is to replace the default, required ROOT web application on the PAS for OpenEdge Tomcat server. This makes the PAS for OpenEdge instance a secured, blank canvas onto which existing ABL applications can be deployed as OpenEdge Application Archive (OEAR) files. For more information about OpenEdge Archive files, see Package an ABL application as an OpenEdge Application Archive.

About licensing and installation

Install OpenEdge on the machine where you want to run the application server. PAS for OpenEdge development mode is installed as part of the **Complete** installation procedure.

For more information about OpenEdge installation, see Install OpenEdge or the Installing Made Easier video.

To run PAS for OpenEdge in production mode, you must obtain a Progress Production Application Server for OpenEdge (`Progress Prod AS for OE`) license. For more information, see Licensing.

When you install OpenEdge, you must select a product installation directory and a separate work directory. The following table provides information about these directories and their defaults:

Default	Windows	Linux	Notes
OpenEdge installation directory	`C:\Progress\OpenEdge`	`/usr/dlc`	Referenced as `$DLC` on Linux and `%DLC%` on Windows.
OpenEdge work directory	`C:\OpenEdge\WRK`	`/usr/wrk`	Must be separate from the OpenEdge installation directory.
PAS for OpenEdge installation directory	`%DLC%\servers\pasoe`	`$DLC/servers/pasoe`	
PAS for OpenEdge instance directory	`C:\OpenEdge\WRK\oepas1`	`/usr/wrk/oepas1`	`oepas1` is the default OpenEdge development instance packaged with installation.

Note: PAS for OpenEdge scripts are located in the `bin` directories of both the PAS for OpenEdge installation (`%DLC%\servers\pasoe\bin`) and each PAS for OpenEdge instance (`C:\OpenEdge\WRK\oepas1\bin`).

About the PAS for OpenEdge configuration tools

OpenEdge provides a number of tools to administer a PAS for OpenEdge instance. These tools include:

- **TCMAN**—A command-line utility for configuring and managing PAS for OpenEdge instances. The tool also enables you to view details such as operating system (OS) and server information, detailed server status, memory leaks, and HTTP session information.

  For more information, see "TCMAN Reference" in *PAS for OpenEdge Administrative Tools Reference*.

- **PASMAN**—A command-line utility that runs as a TCMAN wrapper from the `$DLC/bin` directory. This utility allows PAS for OpenEdge system administrators to manage multiple instances.

- **OEPROP**—A command-line utility used to display and manage properties in the *`instance-name`*`/conf/openedge.properties` file.

- **OpenEdge Management**—A browser-based administration tool that allows you to create and manage PAS for OpenEdge instances. This tool runs as either a Windows or UNIX client of the AdminServer. OpenEdge Management can create and configure, start, stop, and delete a PAS for OpenEdge instance as well as view log files. OpenEdge Management also allows you to monitor various aspects of your PAS for OpenEdge environment. In order to create and manage instances with OpenEdge Management, the `oemanager.war` file must be deployed to the instance.

- **REST APIs**—REST APIs give you visibility into your PAS for OpenEdge environment and let you take appropriate actions based on your findings. REST APIs are enabled by the `oemanager.war` web application and can be viewed through the Swagger UI. For more information, see Use Swagger UI to explore management REST APIs on page 258.

  For the REST API reference documentation, see "REST API Reference for oemanager.war" in *PAS for OpenEdge Administrative Tools Reference*.

- **OEJMX**—A command-line utility that can list all of the MBeans, including attributes and operations for a PAS for OpenEdge instance running on a local machine, and also run MBean queries. OEJMX does everything that deploying the `oemanager.war` web application can do, but without opening access to the instance through the web server. For more information, see Use OEJMX to manage and monitor an instance on page 244.

This content uses the PASMAN command-line utility and OpenEdge Management tools to illustrate the daily tasks of the PAS for OpenEdge system administrator.

Tomcat utilities

PAS for OpenEdge includes the standard Apache Tomcat administrative utilities (`startup`, `shutdown`, `configtest`, `version`, 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. The extended administrative functionality 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.

TCMAN

TCMAN is a command-line utility for administering and managing PAS for OpenEdge. It extends the functionality that is provided by the core Apache Tomcat administrative scripts found in the *`instance-name`*`/bin` folder.

For more information, see "TCMAN reference" in *PAS for OpenEdge Administration Tools Reference*.

Syntax

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

Parameters

`$CATALINA_HOME|$CATALINA_BASE`

`$CATALINA_HOME` is the path of the installation directory of PAS for OpenEdge, typically *`OpenEdge-Install-Dir`*`/servers/pasoe`. Always run TCMAN from `$CATALINA_HOME/bin` when you create new instances or worker configuration files. 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 the directory of an instance.

Note:

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

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

`action`

The TCMAN action to invoke.

`general_options`

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

Type `tcman.sh help` *`action`* for a list of general options that are applicable to a particular action.

`action_options`

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

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 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 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. The PASOESTART command executes until the instance starts without any fatal errors, or until a time-out limit is reached and the startup process is terminated.

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]
[-start] [-halt]
[-query] [-help]
```

Executing PASOESTART from TCMAN:

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

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 does not start fully within the time-out interval, it is considered hung and is stopped.

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

Note: The `-timeout` option is not required if you specify the `-v` option or the `-restart` option.

`-v`

Displays verbose execution information to the console.

`-silent`

Sends execution information to a log file (*instance-name*`/logs/pasoestart.log`).

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

`-json`

Formats completion status information as a JSON object and sends 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`

Stops the instance and performs a full startup if the instance is already running or is in a hung state. If the instance is in the stopped state, then this option has no effect.

`-start`

If the instance is not running, execute this parameter to start the instance.

`-halt`

This parameter stops a running instance.

`-query`

This parameter prints or displays the current state of the instance.

`-help`

This provides the information about the usage of `pasoestart` command.

`-archive`

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

`-nostop`

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

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

Status Codes

PASOESTART is 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. For more information about known state, see Status Information. After PASOESTART completes, the instance's startup completion status is in the OS error variable (`%errorlevel%` for Windows, and `$?` for UNIX). The following table lists and describes the status codes:

Status Code	Description
0	The instance started or 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 or all of it unusable, and is left in the `stopped` state for troubleshooting and problem resolution.
3	The instance is already in the `running` state, was not restarted, and is left in a `running` state that may, or may not, be usable.
4	The instance was started 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 is hung in the `starting` state, was not restarted, and is left in the hung `starting` state.
6	The instance is hung in the `stopping`state, was not restarted, and is left in the hung `stopping` state.
10	Command-line line option errors prevented the 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 sends status information to the console, even when you use the `-silent` option. A full status report includes the 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 PAS for OpenEdge instance at execution time: { `stopped` | `starting` | `stopping` | `running` }.
`Initial processes`	A space-separated list of the OS process IDs for the PAS for OpenEdge instance found at execution time: A single 0 value indicates that there are no OS processes (`stopped`). A single non-zero value indicates that the PAS for OpenEdge server is running but there are no multi-session agents (`starting` or `stopping`). Multiple non-zero values indicate that the PAS for OpenEdge server and at least one multi-session agent is running; however, there may be errors that prevent the instance from serving clients.
`Exit state`	The state of the PAS for OpenEdge instance when the PASOESTART command completes. See the **Initial state** field name entry for state names.
`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 list of PASOESTART exit statuses.
`Exit errors`	A brief list of error or fatal log file entries found in the PAS for OpenEdge log files. This provides hints about why the PAS for OpenEdge instance startup failed and provides references for where the administrator or developer can find the full error output. The Exit errors information may be blank or 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 multi-session agent, and startup event procedures
stopped	stopped	Startup failed and the instance was stopped due to a PASOESTART time-out or errors.
stopped	starting	A startup time-out 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, and then manually stop the instance if the restart is not successful.
started	started	The instance is 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 multi-session agent, and startup event procedures were restarted successfully.
started	stopped	After a PASOESTART `restart`, a running instance was stopped, and failed while restarting due to a time-out 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 `starting`state 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 time-out 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 time-out 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 time-out 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 invokes TCMAN from the PAS for OpenEdge installation directory (*`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`

The TCMAN action to invoke.

For information on actions, see "TCMAN Reference" in *PAS for OpenEdge Administration Tools Reference*.

general_options

One or more of the TCMAN common options that apply to most actions.

Type `tcman.sh help` *`action`* for 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, then the action applies to the server installation directory in `$CATALINA_HOME`. The actions that you can use `-I` with PASMAN to specify a particular instance are:

Action	Description
clean	Cleans up or archives log files.
config	Displays and manages configurations
deploy	Deploys a web application
disable	Stops a web application
enable	Starts a web application
env	Displays runtime environment information
feature	Displays or modifys server features
info	Displays OS and server information
leaks	Displays memory leaks
list	Lists deployed applications
plist	Lists process ids
reload	Reloads a web application
resources	Displays global server resources
service	Registers and manages an instance as a Windows service
sessions	Displays web application HTTP sessions
showproc	Shows Windows process information

Action	Description
start	Starts an instance
status	Displays detailed server status
stop	Stops an instance
test	Tests a server configuration
undeploy	Undeploys a web application
version	Displays server, OS, and run-time version information

Other actions only apply to the PAS for OpenEdge installation 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	Creates an instance
delete	Deletes an instance
instances	Displays server instances
register	Registers an instance for tracking
unregister	Stops tracking an instance
workers	Creates a Tomcat worker configuration file

`action_options`

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

SECPROP

SECPROP is a command-line utility for managing properties in the `oeablSecurity.properties` file.

The oeablSecurity.properties file

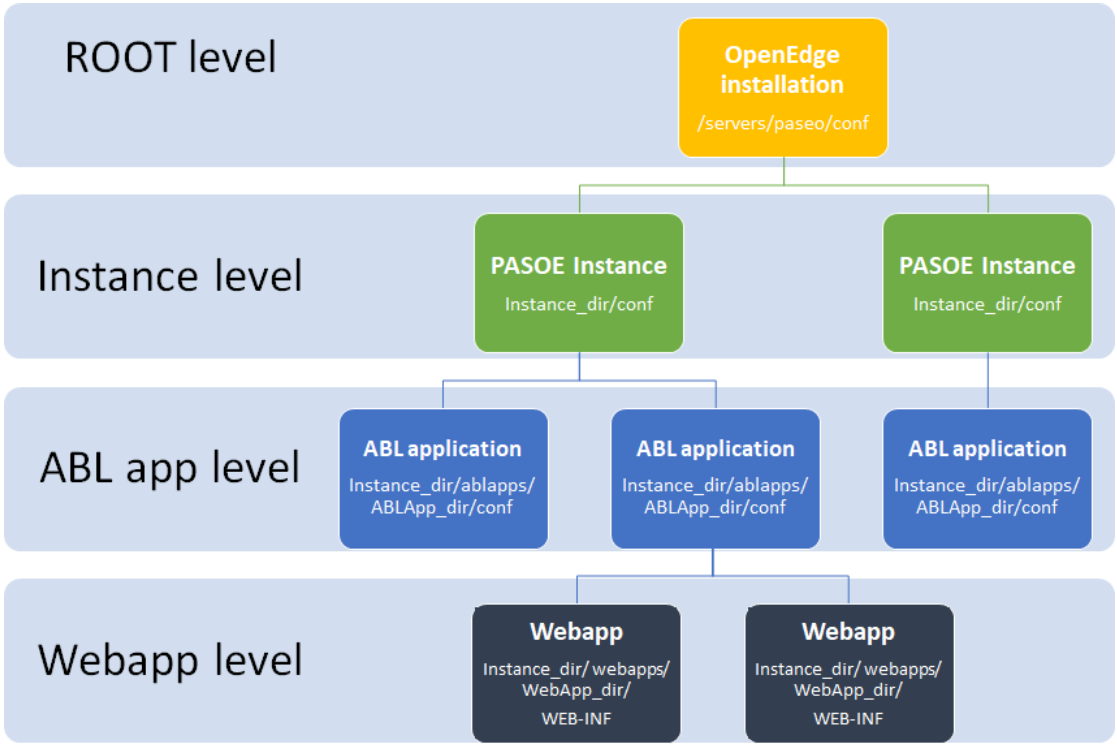
The `oeablSecurity.properties` file provides an easy way for you to configure security properties for a PAS for OpenEdge web application. It acts as an interface to the Spring Security framework that guards the web application. Instead of changing Spring bean constructor and property settings in multiple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

Security property levels

Even though 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 is represented in the following diagram:



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

For example, if you modify the `oeablSecurity` file for a PAS for OpenEdge instance, then 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, then 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 use the SECPROP utility to modify the `oeablSecurity.properties` file at the top-most level; that is, in the *`OpenEdge_installation`*`/servers/pasoe/conf` directory. Furthermore, if you modify properties at any of the child levels, then any changes to the parent-level properties apply only to new children of the same parent.

Syntax

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

Log level

Use the following parameters listed 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 the -v parameter returns a single line log message for each operation performed or error thrown.

`-g`

Sets the logging level to DEBUG. The -g parameter returns debug messages about the operation that was performed or attempted and the files that were accessed. It also includes INFO-level messages.

`-t`

Sets the logging level to TRACE. The -t parameter detailed messages about each method that was called internally while attempting to perform an operation. It also includes DEBUG-level and INFO-level messages.

Operation type parameters

`-h/-help`

Displays the syntax and parameter descriptions for the SECPROP command.

`-m/-manage`

Performs list, update, add, or delete operations.

`-f`

This parameter allows you to merge an existing `oeablSecurity.properties` property file with any level of an instance, such as instance level, ABL application level, and web application level. You must specify `-f` option to use the merge functionality. This functionality simplifies the procedure to add or update properties in the `oeablSecurity.properties` file.

For example, to merge the `myoeablSecurity.properties` file with the `client.login.model=saml` property, perform the following:

Note: The contents of `myoeablSecurity.properties` file should be `client.login.model=saml`.

To merge this file with an instance, type:

```
secprop -f -i oepas1.oepas1.ROOT myoeablSecurity.properties
```

To view the update for the instance, ABL application, and web application, type:

```
secprop -m -iaw oepas1.oepas1.ROOT client.login.model
```

You can view the output as:

```
i: client.login.model=saml
a: client.login.model=anonymous
w: client.login.model=anonymous
```

`-r`

Resolves a property value across hierarchical levels. The SECPROP utility attempts to find the property in the `oeablSecurity.properties` file for a specified web application, ABL application, or PAS for OpenEdge instance. If the utility does not find a properties file at the specified level, it searches 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, then an error is displayed.

Filters

`-c`

List the properties in the master `oeablSecurity.properties` file.

`-i` *`PASOE_instance_name`*

The name of a PAS for OpenEdge instance, for example, `-i oepas1`. The operation that you perform (listing, updating, adding, or deleting security properties) applies only to this PAS for OpenEdge instance.

`-a` *`PASOE_instance_name.ABLApp_name`*

The name of an ABL application, for example, `-a oepas1.myABLApp`. The operation that you perform (listing, updating, adding, or deleting security properties) applies only to this ABL application.

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

The name of a web application, for example, `-w oepas1.myABLApp.myWebApp`. The operation that you perform (listing, updating, adding, or deleting security properties) applies 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`. You can define operations (listing, updating, adding, or deleting security properties) that apply to the PAS for OpenEdge instance and the ABL application.

`-iaw PASOE_instance_name.ABLApp_name.Webapp_name`

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

Read operations

To read the value of a property, 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, for 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 check whether 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
```

To find the value of different properties, use the `-r` operational type parameter, for example:

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

Update operations

To update the value of a property, 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, for 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, then the update operation fails for that file and the SECPROP utility generates 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, then the add operation fails for that file, and the SECPROP utility generates 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, for example:

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

Performing multiple operations

You can perform multiple operations in the same SECPROP command, using the relevant parameters and switches, for example:

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

OEPROP

OEPROP is a command-line utility for displaying and managing properties in the *instance-name*/conf/openedge.properties file.

Note: Use TCMAN (tcman config) to manage the properties in *instance-name*/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`

Displays the syntax and parameter descriptions for the oeprop command.

group_name.prop_name

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

group_name.prop_name=value

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

+group_name.prop_name=value

Adds a new property to the specified group in .../conf/openedge.properties and sets its value.

~group_name.prop_name

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

~group_name

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

-R

Removes the non-empty groups when used with *~group_name*.

-f *filename*

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

This option is particularly useful when you deploy and configure new web applications. It allows you to add property names and property groups in one step.

-v

Displays verbose information when OEPROP runs.

-g

Displays debug information when OEPROP runs.

-t

Displays 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

The `paspropconv` tool migrates 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. It is important 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 when 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`: Use the file to create the comments section of the `pasoename.ubrokername.oemerge` file. Currently, 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 release 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 *Migrate Classic AppServer Applications to PAS for OpenEdge*.

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 Classic 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, then specify the name of your PAS for OpenEdge instance.

`--pasoeWebAppName`

(Optional) 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 is migrated to the ROOT ABL web application.

`--pasoeMergeFile`

(Optional) 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 is named `pasoeAppName.ubrokername.oemerge`.

`--pasoeSetEnvFile`

(Optional) 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 `pasoeAppName_setenv.sh/.bat` (for UNIX/Windows).

`--convNotesDBFile`

(Optional) Specify the name of the file. This file is generated on running `paspropconv`. By default, the generated file will be named `paspropconv_notesdb.en`.

`--logFile`

(Optional) Specify the name of the `paspropconv` tool's log. By default, the log is named `paspropconv.log`.

`--loggingLevel`

(Optional) 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`

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

`--mergepropExe`

(Optional) Specify the full pathname of the `mergeprop` utility, if you are not using the default location (*`oe-install-dir`*`/bin/mergeprop.bat`). The default value is `mergeprop`.

`--help`

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

Utilities to manage keys and digital certificates

The following OpenEdge utilities, typically run from the Proenv command-line environment, 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:

  See the `stspwdutil` utility in *OpenEdge Getting Started: OpenEdge Authentication Gateway Guide*. `stspwdutil` has 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. This type of a certificate can be authorized by your 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 TLS 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 "Manage OpenEdge Keys and Certificates" in *Manage OpenEdge Keys and Certificates*.

.

OpenEdge Management and OpenEdge Explorer

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

For more information, see Introduction to OpenEdge Management.

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.

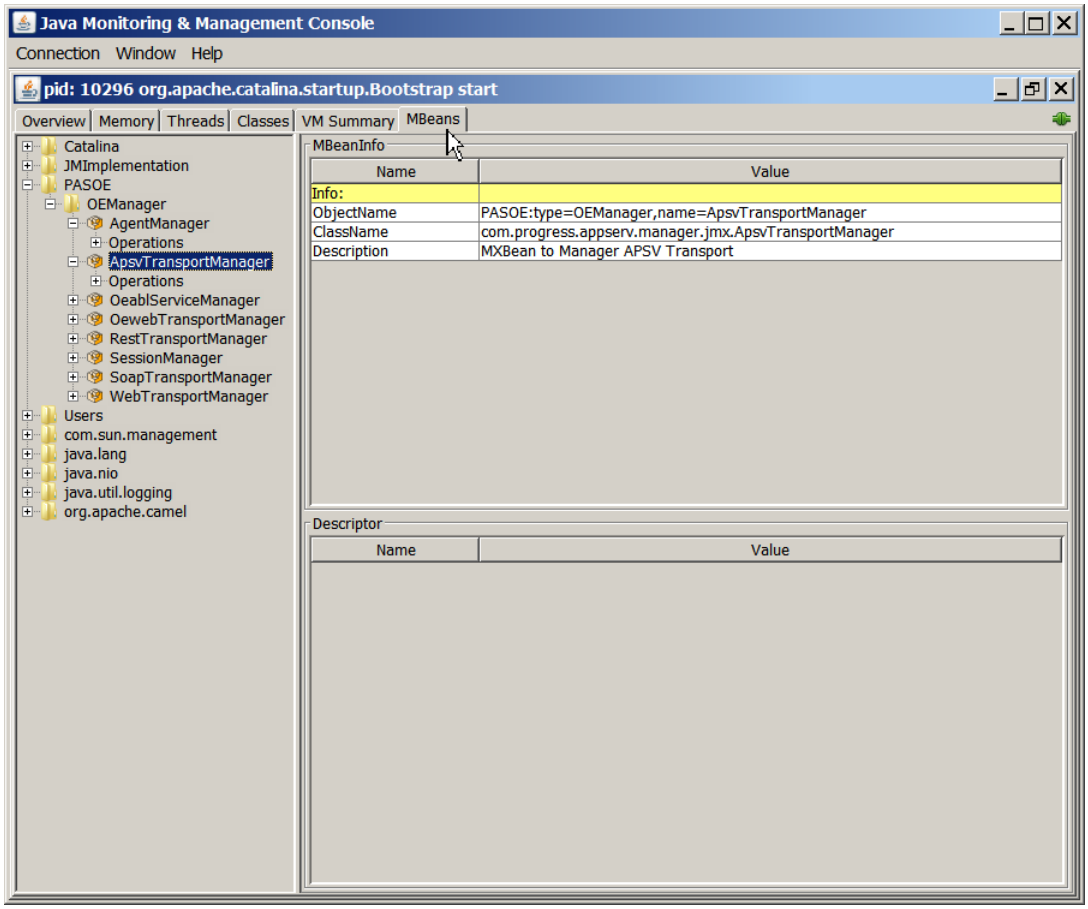
You should not use JConsole on a local production server because it uses significant resources. Using JConsole with a remote connection to a production server avoids this problem. However, enabling remote connections requires more attention to security since it opens the door to unauthorized access to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

For more information, see Run JConsole locally on page 40 and Run JConsole remotely on page 42.

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.

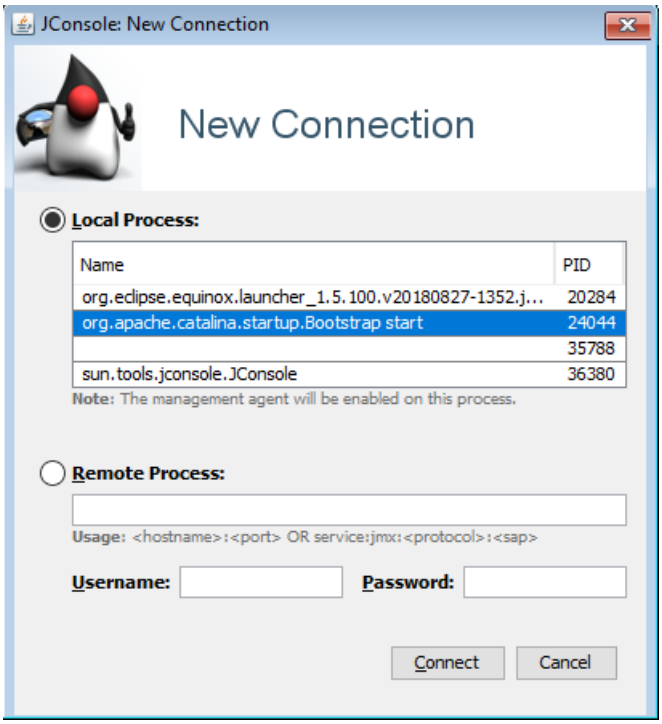
The output of OEJMX requests and queries is written to a file. For more information, see OEJMX utility on page 43 and Use OEJMX to manage and monitor an instance.

Run 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.

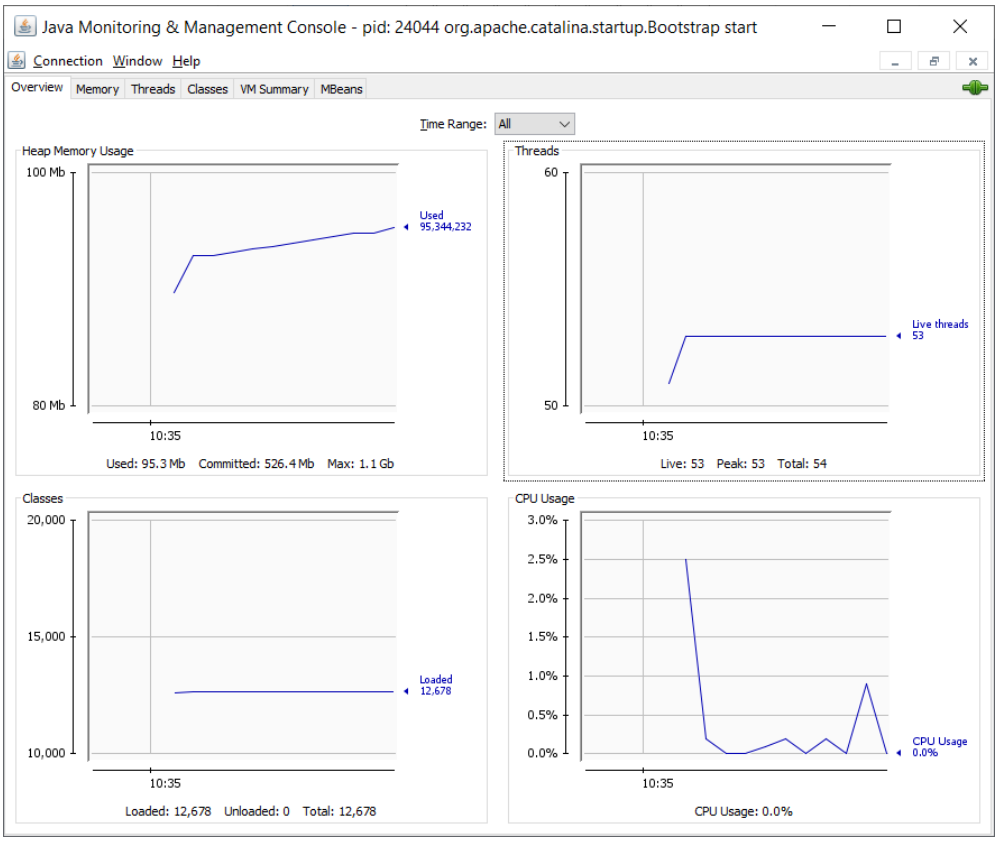
1. Open a command-line console, navigate to the JDK **bin** directory (for example `cd C:\Program Files\Java\jdk1.8.0_101\bin`), and enter `jconsole.exe`.

The JConsole New Connection 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.



Run 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.

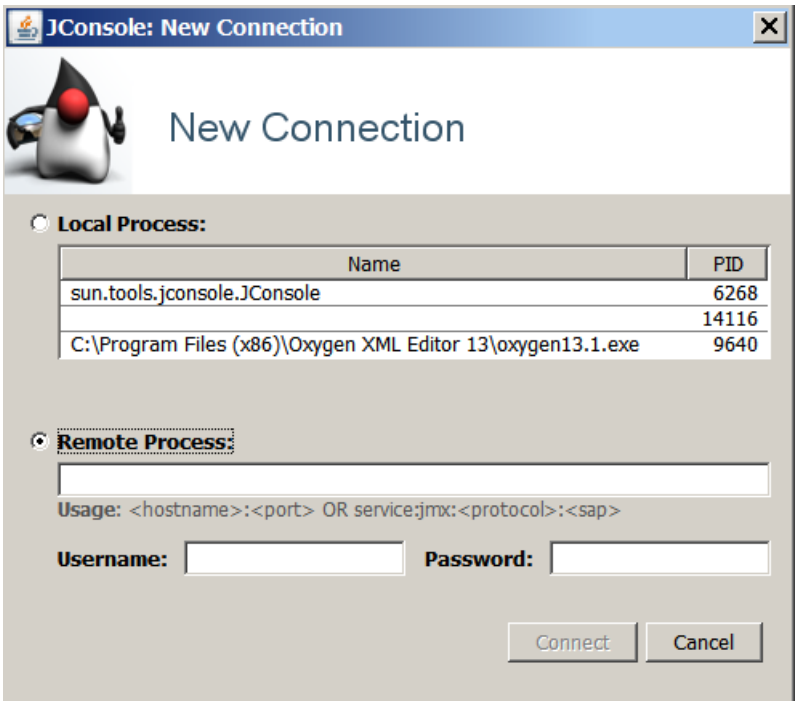
   Important: `jmxremote.password` is a plain-text file that contains the username/password combination that you need in order to log in to the instance from a remote JConsole. The default username and password (`jmxadmin`, `jmxadmin`) should be changed for security reasons.

   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.

OEJMX utility

JMX (Java Management Extensions) is a Java framework-supplied technology that is part of the core JVM process. The Java framework, Tomcat (which is a Java application), and OpenEdge make extensive use of JMX for working with their internal management and configuration. For more information about JMX in OpenEdge, see JConsole and JMX on page 39.

OEJMX is an OpenEdge command-line interface that facilitates automation scripting by administrators to get, set, and execute properties and actions within Tomcat and ABL web applications. You can use OEJMX to:

- List all the JMX 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.

For usage information, see Use OEJMX to manage and monitor an instance.

Syntax

```
oejmx.{sh|bat} [ -C ] [ -Q query_parameter ] [ -O output_file_name ][ -R ]
```

Parameters

-C

Generates a list of available bean operations and attributes. The default output location for the query results is *CATALINA_BASE*\temp\beanInfo-{TIMESTAMP:yyMMdd-HHmm}.txt, but the output location can be specified using the -O parameter.

-Q query_parameter

Specifies the path to a file with a set of JMX queries or a single JMX query. For more information about writing JMX queries, see Use OEJMX to manage and monitor an instance on page 244.

There is an example JMX queries file called default.qry in the *CATALINA_BASE*\bin\jmxqueries directory. To call this query from the *CATALINA_BASE*\bin directory, enter the following command:

```
proenv>oejmx -Q jmxqueries/default.qry
```

A single JMX query is a JSON string with escaped quotation marks, as shown in the default.qry file. For example:

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

When using a single JMX query, the result prints to the console. For example:

```
proenv>oejmx -Q
{\"O\":\"PASOE:type=OEManager,name=AgentManager\",\"M\":[\"getAgents\",\"oepas1\"]}
{"getAgents":{"agents":[{"agentId":"_lPkhGI8T_-INocNySRdjA","pid":"9508","state":"AVAILABLE"}]}}
```

Note: When using a single JMX query on the command line, tokens are not supported. Tokens are only supported in query files. Tokens are special inputs that take the place of a specific value, such as an application name or agent ID. You can use tokens in your JMX queries to get information for all running ABL applications, web applications, agents or sessions. The supported tokens are APP_NAME, AGENT_ID, AGENT_PID, SESSION_ID, and WEB_APP_NAME, but are only valid for certain queries. For more information, see the OEJMX Query Reference topics.

Note: This parameter is ignored if the -C attribute is used.

-O output_file_name (with path)

Specifies the output filename and path, and may specify a timestamp when used with the following templates: {TIMESTAMP} or {TIMESTAMP:*Java-datetime-format*}

Default values:

- If the -C argument is used:
  CATALINA_BASE\temp\beanInfo-{TIMESTAMP:yyMMdd-HHmm}.txt
- If the -C argument is not used:
  CATALINA_BASE\temp\query-out-{TIMESTAMP:yyMMdd-HHmmss}.txt

For example, the following command produces a file in the *CATALINA_BASE*\temp directory called queryname-20191126-112127.txt:

```
proenv>oejmx -Q jmxqueries/default.qry -O
C:\OpenEdge\WRK\oepas1\temp\queryname-{TIMESTAMP:yyMMdd-HHmmss}.txt
```

Note: This parameter is ignored if a single JMX query is used.

-R

Causes the query results only to be written to the output file. Otherwise, the output file also includes the query texts.

Note: This parameter is ignored if the -C argument or a single JMX query is used.

Before you move to production

Configure the staging environment identically to the production environment, and then install OpenEdge.

Developers working with PAS for OpenEdge development licenses can provide whole, compressed instances, or individual applications and services, which system administrators can then deploy to the staging environment.

System administrators should then apply production configurations as if the instance was running in a live user environment. At this stage, the system administrator can do regression testing, functional testing, performance testing, and tuning. The testing ensures that when the instance goes to production, the system administrator has a better understanding of the run-time performance and configuration.

Production checklist

The following topics describe configuration and other issues that you should know about in order to run the production version of PAS for OpenEdge:

Topic	Description
Configuration tools	Familiarize yourself with the instance configuration tools.
Missing and disabled features in production	Understand the features that are not enabled by default in production instances.
Create a PAS for OpenEdge production instance	Create your PAS for OpenEdge production instance.
Check the security model of the server	Ensure that your instance is running with the production security model.
Configure the instance for production	Complete additional configuration tasks such as Set the PROPATH, Set database connections, Enable transports, and Set agents and sessions.
User and file permissions	Set user access controls.
Enable authentication	Set the authentication level for your instance.
Secure an instance	Learn about PAS for OpenEdge security considerations and best pratices.
Set environment variables	See this topic if you want to set the same environment variables each time an instance is launched.

2

Create and configure PAS for OpenEdge instances

As a system administrator you create and configure Progress Application Server for OpenEdge (PAS for OpenEdge) instances in production mode. There are different ways to create and configure server instances. This content describes using the PASMAN command-line utility and the OpenEdge Management graphical user interface to perform these tasks.

For details, see the following topics:

- Create a PAS for OpenEdge production instance
- Start and stop a PAS for OpenEdge instance
- Automate lifecycle event scripts
- Configure a PAS for OpenEdge instance
- Connect to an OpenEdge database from PAS for OpenEdge
- Register a PAS for OpenEdge instance as a Windows service
- Configure private PAS for OpenEdge instance tracking
- Delete a PAS for OpenEdge instance

Create a PAS for OpenEdge production instance

Creating a PAS for OpenEdge instance involves defining a name and location for the instance as well as defining various configuration properties.

With the development mode of PAS for OpenEdge, a default instance, `oepas1`, is created in your work directory. However, a default instance is not created when you install a production version of PAS for OpenEdge. A default instance would mean that ports, name spaces, and other values that are well-known would be exposed, and having well-known values is a security risk. Therefore, before you deploy an application to PAS for OpenEdge, you must create an instance.

When you create an instance, the properties of the instance are automatically set to the directory where the new instance is created and the related files and logs are stored. The following properties are set when the new instance is created and can be controlled by the system administrator:

- The environment variables and paths for running commands
- The HTTP and HTTPS transports, and ports
- The username and password for the instance
- The type of instance (development or production)

See also:

- Use PASMAN command-line utility to create an instance
- Use OpenEdge Management to create an instance

Use PASMAN to create an instance

OpenEdge is shipped with a default PAS for OpenEdge development instance located in your `OpenEdge/WRK` directory named `oepas1`. In the production version of PAS for OpenEdge, a default instance is not provided, and you must create one.

To use the PASMAN command-line utility to create a new PAS for OpenEdge instance:

1. Run the Proenv utility.
2. Create an instance. For example:

   ```
   pasman.[bat|sh] create -Z prod -p 9910 -P 9911 -s 9912 instance-path/instance-name
   ```

   - On UNIX, the only required parameters are the filepath and instance name.
   - On Windows, a shutdown port, set by the `-s` option, is also required.
   - The `-Z prod` switch defines the security model as production.
   - The `-p` option specifies the HTTP port. The default is 8810.
   - The `-P` option specifies the HTTPS port. The default is 8811.
   - The defaults can be set during the OpenEdge installation.

     You can find additional configuration parameters in "TCMAN Reference" in *PAS for OpenEdge Administration Tools Reference* and PASMAN on page 25.

Use OpenEdge Management to create an instance

OpenEdge Management provides a sample PAS for OpenEdge instance named `oepas1` with a default configuration. Each new PAS for OpenEdge instance created with OpenEdge Management uses the default configuration. You can edit the PAS for OpenEdge configuration properties to meet your configuration requirements.

To run OpenEdge Management in a production environment locally, you must have the OpenEdge Manager and Tomcat Manager web applications deployed on your PAS for OpenEdge instance.

Note: OpenEdge Management can be created remotely. For more information, see Introducing OpenEdge Explorer and OpenEdge Management.

To create a PAS for OpenEdge instance using OpenEdge Management:

1. Navigate to **Resources**>**New**>**Progress Application Server**.
2. Specify the following OpenEdge configuration properties:

Property	Description
Instance Name	A name for the PAS for OpenEdge instance. This property is mandatory. Instance name is case sensitive. It can include any character except periods (.) or brackets ([ ]). The name must be unique among all configured PAS for OpenEdge names.
HTTP Port	An unused port number associated with the PAS for OpenEdge HTTP. By default, this is set to 8810.
HTTPS Port	An unused port number associated with the PAS for OpenEdge HTTPS. By default, this is set to 8811.
Shutdown Port	An unused port number for shutdown. If you are creating a PAS for OpenEdge instance on a Windows machine, this property is mandatory.

3. Click **Create**.

The new PAS for OpenEdge instance is listed with other resource instances on the **Resources** page in OpenEdge Management.

Check the security model of the instance

To verify which product security model (development, production, or pas) the instance is running:

1. Run the Proenv utility.
2. Perform one of the following tasks:
   - Get the environment details by running the `pasman env` command on the instance. For example:

     `pasman.[sh|bat] env -I instance-name`

     The security model is listed as one of the last properties in the output.
   - To list only the security model of the instance, run the following command:

     `pasman.[sh|bat] config -I instance-name psc.as.security.model`

     The security model property is displayed.

You cannot change a development security model to a production model, but you can create a new instance with the `-Z prod` switch that is described in the Use PASMAN to create an instance topic.

OpenEdge 12.2 offers a third security model, which is a subset of the production security model, that can be specified at instance creation using the `-Z pas` switch. For more information, see Security models.

Start and stop a PAS for OpenEdge instance

After you create a PAS for OpenEdge production instance but before you can deploy applications to the instance, you must start the instance.

See also:

- Use PASMAN to start and stop an instance
- Use OpenEdge Management to start and stop an instance

Use PASMAN to start and stop an instance

To start or stop a PAS for OpenEdge instance using the PASMAN command-line utility:

1. Run the Proenv utility.
2. Run the `pasman.[bat|sh] pasoestart` command:

   ```
   pasman.[bat|sh] pasoestart -I instance-name -timeout 200
   ```

   The `pasoestart` command produces a more stable and predictable startup process by performing the start actions, and then the error checking and process validation. It executes until the instance starts without any fatal errors, or until a timeout limit is reached and the startup process is stopped.
3. Run the `pasman.[bat|sh] env` command to retrieve system information such as the HTTP port:

   ```
   pasman.[bat|sh] env -I instance-name
   ```

4. Test that the server is running by entering the following URL into a web browser:

   ```
   http://localhost:port
   ```

   The *port* is the HTTP port that you set when creating the instance and is listed by the `pasman env` command. Use `localhost` for local instances, or enter the name of the remote host.
5. Shut down the server by running the `pasman.[bat|sh] stop` command:

   ```
   pasman.[bat|sh] stop -I instance-name
   ```

Use OpenEdge Management to start and stop an instance

To start or stop a PAS for OpenEdge instance using OpenEdge Management:

1. On the **Resources** page in OpenEdge Management, select the instance you want to start or stop.

   The Progress Application Server: `instance-name` page appears.
2. Select **Start** or **Stop**.

Run PAS for OpenEdge as a non-root user

On UNIX systems, you can start a PAS for OpenEdge instance as a non-root user if the following conditions exist:

- You can be authenticated by the operating system.
- OpenEdge executables and scripts have the execute privilege and file permissions that grant you the required read and write access operations that are part of the PAS for OpenEdge startup process.

Operating system authentication requirements

The PAS for OpenEdge instance must be started with an account that has the privileges that are needed to validate user accounts and passwords on that particular UNIX system's configuration. To authenticate the user who is attempting to start a PAS for OpenEdge instance, the instance uses the UNIX system's password files, which may be located in either the NIS password map or the `/etc/passwd` file, or in both locations depending on the system configuration, as follows:

- If only account validation is performed, the UNIX system looks up that account to verify that it exists and does not use the shadow password file (`/etc/shadow`).
- If both account validation and password validation are performed, the UNIX system uses both the `/etc/passwd` files and `/etc/shadow` files, if the system is configured by the system administrator to use them.

UNIX protects the `/etc/passwd` and `/etc/shadow` files by requiring a process to have root privileges to access them when validating passwords. However, the NIS password map generally does not require root privileges to access accounts and validate passwords. The level of security is configured by the UNIX system administrator. To determine whether you require root privileges to start PAS for OpenEdge, you may have to consult the UNIX system administrator.

Requirements for changing file ownership and access permissions

After you install OpenEdge on a UNIX system, you typically configure file permissions in accordance with your company's policies. Note the following with regards to the permissions that are required for changing file ownership and access mode on OpenEdge files:

- The OpenEdge and OpenEdge Management product installations must be performed by a root user.
- If you are enabling OpenEdge Management (`fmconfig -enable`) or remote instance containers (`fmconfig -enable -host` *`host-name`*), you must run the commands as root otherwise the commands will fail.
- If you are using OpenEdge Management, the commands to change file ownership and access should be changed only after you have enabled OpenEdge Management or remote containers. Otherwise some files may revert back to only having root permissions and therefore could result in the Progress DBA being unable to start the PAS for OpenEdge instance or OpenEdge Management.
- After installation is complete, but prior to changing any file permissions, make sure that all Progress processes are stopped.
- With certain OpenEdge Management versions, the `chgrp` and `chmod` commands fail when run against the `fathom.init.params` file and `/var/tmp/Jetty__9090__*` directories because they might not exist. You can ignore the errors.
- All commands to change file ownership or access mode (that is, the `chgrp` and `chmod` commands, respectively) must be performed by a root user.
- The `progress` group should be used only for Progress DBA users. Other, non-DBA users should not be part of this group. Progress clients should be part of a different group; for example, `progusers`, and the database files should have `progusers` as its group and have `g+rw` permissions associated with them. The permissions of the directory containing the database files should also have the group `progusers` and `r-x` permissions (that is, no write permissions) to prevent someone from deleting the files. All other permissions should be removed from the database directory and files, using the `chmod o-rwx` *`db-files`* command. The Progress DBA should be a member of both the `progress` and `progusers` groups by comma-separating them within the `/etc/groups` file.
- Progress recommends that you use a script to apply the file permissions so that the process can be replicated; for example, after installing OpenEdge on multiple host machines or applying a service pack.

Steps to change file ownership and access

The steps provided in this section use the following names and variables:

- `progress` is the group that the non-root user (for example, the DBA) will be a member of.
- `dlc` is the directory where OpenEdge is installed.
- `wrk` is the OpenEdge working directory.
- `oemgmt` is the OpenEdge Management installation directory.
- `wrk_oemgmt` is the OpenEdge Management working directory.

Also, the steps described in this section make the following file ownership and access changes:

- Recursively change the group of all directories and files within the `dlc`, `oemgmt`, `wrk`, and `wrk_oemgmt` directories to `progress`.
- Add `-rwxrwx---` permissions for all directories, including those within the recursive (`-R`) `chmod` commands.
- Add `-rw-rw----` permissions for all files, including those within the recursive `(-R) chmod` commands. The use of uppercase `X` with `chmod` sets `x` permissions for directories but not for the files within.
- Recursively remove `--------w-` permissions for all files and directories in the `dlc` and `oemgmt` directories. (Remember to change the group `progress` and the directory names to suit your configuration.

To change file ownership and access for OpenEdge when OpenEdge Management is not also installed, enter the following commands:

1. `chgrp -R progress dlc`
2. `chgrp -R progress wrk`
3. `chmod -R o-w dlc`
4. `chmod -R ug+rwX dlc/properties ; chmod -R o-rwx dlc/properties`
5. `chmod -R ug+rwX wrk`

To change file ownership and access for OpenEdge when OpenEdge Management is installed and configured after having run `fmconfig -enable` as root, enter the following commands:

1. `chgrp -R progress dlc`
2. `chgrp -R progress wrk`
3. `chgrp -R progress wrk_oemgmt`
4. `chgrp -R progress oemgmt`
5. `chgrp -R progress /var/tmp/Jetty__9090__*`
6. `chmod -R o-w dlc`
7. `chmod -R o-w oemgmt`
8. `chmod -R ug+rwX dlc/properties ; chmod -R o-rwx dlc/properties`
9. `chmod -R ug+rwX wrk`
10. `chmod -R ug+rwX wrk_oemgmt ; chmod -R o-rwx wrk_oemgmt`
11. `chmod -R ug+rwX oemgmt/MQ6.1/Fathom1 ; chmod -R o-rwx oemgmt/MQ6.1/Fathom1`
12. `chmod -R ug+rwX oemgmt/MQ6.1/SonicMQStore ; chmod -R o-rwx oemgmt/MQ6.1/SonicMQStore`
13. `chmod -R ug+rwX oemgmt/MQ6.1/log ; chmod -R o-rwx oemgmt/MQ6.1/log`
14. `chmod -R ug+rwX oemgmt/db ; chmod -R o-rwx oemgmt/db`
15. `chmod -R ug+rwX oemgmt/etc ; chmod -R o-rwx oemgmt/etc`
16. `chmod -R ug+rwX oemgmt/jspwork ; chmod -R o-rwx oemgmt/jspwork`

17. `chmod 770 oemgmt/web`
18. `chmod 770 oemgmt/config`
19. `chmod 770 /var/tmp/Jetty__9090__*`
20. `chmod 660 dlc/fathom.init.params`

Automate lifecycle event scripts

Tomcat servers have the ability to start 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 shuts down the third-party server when the instance shuts down. The scripts must be located in the *`instance-name`*`/bin` directory and use the following naming conventions.

Table 1: Automated lifecycle scripts

Script	Lifecycle event
`scriptPrefix``_startup.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``_shutdown.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 that the Tomcat startup/shutdown process waits for the event script to complete before continuing. If a script hangs, then it prevents the lifecycle event from completing.

Logs and errors

The standard server logs, located in the *`instance-name`*`/logs` directory, track when the scripts are called. If errors are raised by the script, then 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-name`*`/logs` directory.

Configure a PAS for OpenEdge instance

System administrators can view PAS for OpenEdge configuration properties and make changes using the OpenEdge Management tool as well as the PASMAN command-line utility.

In most cases, you do not need to modify the values of the properties contained in the properties files. However, if required, you can modify the existing property values or add new properties. Depending on the property file that you want to edit, use one of the following tools:

- `pasman config`—View or modify configuration properties.
- `pasman feature`—Enable or disable server features.
- OpenEdge Management—Edit the configuration files using a graphical user interface.

See also:

- Use PASMAN to configure an instance
- Use OpenEdge Management to configure an instance

Use PASMAN to configure an instance

The PASMAN utility can be used to view, add, update, or delete instance properties.

When you run `pasman.[bat|sh] config` with no parameters, it displays the core configuration properties of the Tomcat server including the `appserver.properties`, `jvm.properties`, and `catalina.properties` located in the *`instance-name`*`/conf` directory.

Note: You can only view `jvm.properties`. You cannot modify its contents with the `pasman config` action.

To change property values:

1. Run the Proenv utility.
2. Enter the `pasman config` command to display the properties of your instance:

   ```
   pasman.[bat|sh] config -I instance-name
   ```

3. Enter a specific property name to see the value of a property, for example:

   ```
   pasman.[bat|sh] config psc.as.http.port -I instance-name
   ```

   This command returns the current value of the HTTP port for the specified instance.

4. Update the value of the property:

   ```
   pasman.[bat|sh] config psc.as.http.port=6543 -I instance-name
   ```

5. Run the `pasman config` command on the property again, to ensure it was updated:

   ```
   pasman.[bat|sh] config psc.as.http.port -I instance-name
   ```

6. To add a new property and value pair to the `appserver.properties` file, type the following:

   ```
   pasman.[bat|sh] config +my.example.dir=/home/example -I instance-name
   ```

7. To remove a property and value pair from the `appserver.properties` file, type the following:

   ```
   pasman.{bat|sh} config ~my.example.dir -I instance-name
   ```

Note: All property names are case sensitive.

Use OpenEdge Management to configure an instance

If the `oemanager.war` web application is deployed to your instance, then you can edit the configuration properties of a PAS for OpenEdge instance using OpenEdge Management.

To edit PAS for OpenEdge properties using OpenEdge Management:

1. Navigate to **ABL Application Configuration: Resources** > **Progress Application Server** > ***Instance-Name*** > **ABL Applications** > ***ABL-application-name*** > **Configuration**
2. The Configuration page appears with the following tabs:
   - **Startup Parameters and Environment**—Configure the instance working directory, agent startup parameter and PROPATH settings, as well as other agent and session procedures.
   - **Log Settings**—Configure logging settings for the agent.
   - **Performance Tuning and Limits**—Specify the number of agents, sessions, and timeouts. For more information, see Tune PAS for OpenEdge instances.
   - **Network Environment**—Enable TLS and other network settings.
3. Click **Save** to save your changes.

Connect to an OpenEdge database from PAS for OpenEdge

The OpenEdge Relational Database Management System (RDBMS) network connection policy allows for one network connection per ABL session in the PAS for OpenEdge multi-session agent. Therefore, the application code and PROMON monitoring are the same for both servers.

However, the self-service RDBMS connection policy works differently. Self-service connections are a shared resource in PAS for OpenEdge. Each multi-session agent creates a single self-service connection that shows up in PROMON as its own user connection. Therefore, each ABL session that executes with a database connection parameter (`-db`) or that executes an ABL `CONNECT` statement shares the self-service connection from its managing multi-session agent, and shows up on PROMON with its own user connection. Each ABL session user connection is specific to that session. A single ABL session user connection does not bind the user connection of any other ABL session.

For information about database management, see "Database Administration" in *Manage the OpenEdge Database*.

There are two type values that are specific to PAS for OpenEdge in the PROMON display:

- `SELF/PASA`—A multi-session agent's shared self-service database connection
- `SELF/PASN`—Individual ABL session users that share the multi-session agent connection

OpenEdge ships with a sample OpenEdge database in the `$DLC` directory called `sports2020`.

To configure and verify the database connection for a PAS for OpenEdge instance using the command-line utilities provided by OpenEdge:

1. Run the Proenv utility.
2. Go to your OpenEdge work directory, for example:

```
cd C:/OpenEdge/WRK
```

3. Create a folder to store your database:

```
mkdir database
```

4. Run the following command in Proenv to copy the Sports2020 database that ships with OpenEdge into your work directory:

```
$DLC/bin/procopy $DLC/sports2020
    path-to-openedge-work-directory/database/sports2020
```

Note: $DLC is the UNIX-style environment variable syntax for the directory of your Progress OpenEdge installation. On Windows, use %DLC%.

5. Change directory to *path-to-openedge-work-directory*/database.
6. Start the database server:

```
proserve sports2020
```

7. Change to the /bin directory of your PAS for OpenEdge instance, for example:

```
cd path-to-openedge-work-directory/oepas1/bin
```

8. Use the OEPROP utility to specify the database connection in the Agent Startup Parameter property. The following command shows how to configure a connection to the sports2020 database:

```
oeprop.{bat|sh} AppServer.SessMgr.agentStartupParam="-db
   openedge-install-path/WRK/database/sports2020"
```

This command tells the instance to connect to sports2020 database when the agent starts up.

9. Save the following code to your *path-to-openedge-work-directory*/oepas1/openedge directory to make it available to your PROPATH as server.p:

```
DEFINE OUTPUT PARAMETER custname AS CHARACTER.

FIND FIRST customer.

ASSIGN custname = name.
```

10. Restart your instance using the `pasman pasoestart` command, for example:

```
pasman.{sh|bat} pasoestart -restart -I oepas1 -timeout 200
```

11. Go to the OpenEdge working directory.
12. Run the `prowin` (Windows) or `_progres` (UNIX) command to open the Procedure Editor.
13. Run the following ABL code in the Procedure Editor to test the database connection:

```
DEF VAR appHandle AS HANDLE.
DEF VAR custname  AS CHARACTER.
DEF VAR cnctParam AS CHARACTER.
DEF VAR ret       AS LOGICAL.

CREATE SERVER appHandle.

cnctParam = "-URL http://hostname:port/apsv -sessionModel Session-Managed".

ret = appHandle:CONNECT(cnctParam).

IF NOT ret THEN
        DO:
           MESSAGE "Server not connected".
           RETURN.
        END.
ELSE
        DO:
          MESSAGE "Connected to PAS for OpenEdge!".
          RUN server.p ON appHandle (OUTPUT custname).
          MESSAGE "First customer is: " custname.
          appHandle:DISCONNECT().
                  MESSAGE "Disconnected from PASOE".
        END.

QUIT.
```

Note: Modify the `cnctParam` with the hostname and port of your instance.

This code displays the first customer name in the Sports2020 database. You can save this procedure as `client.p` if you want to use this procedure as a reference.

Register a PAS for OpenEdge instance as a Windows service

To install a PAS for OpenEdge instance as a Windows service, you must have administrator privileges.

A service is an application without a user interface that runs in the background and provides core operating system functionality. Web servers like PAS for OpenEdge and Tomcat typically run as Windows services.

Note: If you run a PAS for OpenEdge instance with the `pasman.[sh|bat] start` command, then 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.

To register and run a PAS for OpenEdge instance as a Windows service:

1. Run the Proenv utility.
2. Run the `pasman service` command. Specify the instance name and the `register` parameter:

   ```
   pasman.[sh|bat] service -I instance-name register
   ```

3. Run the `pasman service` command with an instance name and the `start` parameter.

   ```
   pasman.[sh|bat] service -I instance-name start
   ```

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

Configure private PAS for OpenEdge instance tracking

PAS for OpenEdge development and test users can keep a private list of static PAS for OpenEdge instance alias names, so that each user can use the same alias name of the instance but point to a private file system space unique to each user.

It is expected that ABL developers and QA teams may share the same PAS for OpenEdge product installation and use the same PAS for OpenEdge instance names, but own private copies of the instances. In this way, concurrent development and testing may occur without interference from another user or process of the same user.

PAS for OpenEdge's architecture is comprised of a single, non-runnable product installation (`$DLC/servers/pasoe`) that supports one or more run-time server instances (located in `$WRKDIR`).

A PAS for OpenEdge instance is an Apache Tomcat and OpenEdge directory tree that contains the instance's configuration files, local administration utilities, and Java and ABL web applications. Each created PAS for OpenEdge instance has an alias name, which is a short, human-readable name that can be used to start, stop, administer, register, and unregister, and delete the instance without having to know its exact location in the OS's filesystem.

The installed PAS for OpenEdge product uses a tracking file, known as an instances list (default location: `$DLC/servers/pasoe/conf`), to maintain a list of known functional PAS for OpenEdge instances and their location in the OS's filesystem. When a PAS for OpenEdge instance's alias name is used in the TCMAN or PASMAN administration utility, TCMAN looks-up and uses the OS filesystem path automatically to execute an action.

At installation, PAS for OpenEdge creates an instances list file that provides all OpenEdge installation users with a single shared view of all known PAS for OpenEdge instances. This is the normal mode of administration, where multiple administrators share responsibility for all known PAS for OpenEdge instances. However, there are times when multiple OpenEdge installation users need to create, configure, and run private PAS for OpenEdge instances that are not visible (or known) to any other user. Such may be the case where multiple ABL application developers share a single OpenEdge installation, but do not want other developers to have access to their PAS for OpenEdge instances. Another case may be when running automated ABL application testing where many tests, each with its own PAS for OpenEdge instance of the same alias name, are executing concurrently.

Note: Private PAS for OpenEdge instance tracking is not supported by OpenEdge Management or Progress Developer Studio.

Configure PAS for OpenEdge to enable private instance tracking

The TCMAN and PASMAN administration utilities found in the PAS for OpenEdge product installation can work with PAS for OpenEdge instance alias names if they find an instances file.

When the TCMAN (and PASMAN) administration utility initializes, it looks for an instances list file containing the known PAS for OpenEdge instances. It first looks for the declaration of a private PAS for OpenEdge Instance Tracking file location in the user's environment, and if a declaration is not found, it uses the default shared instance list located in the PAS for OpenEdge product's installation.

Using private PAS for OpenEdge Instance Tracking requires that the user manually add configuration settings in the private account space where the TCMAN utility can find and use it. The user is the one who makes the decision to use Private PAS for OpenEdge Instance Tracking, and where their private instances list file will be located.

PAS for OpenEdge Instance Tracking File Resolution

The TCMAN (and PASMAN) administration utility's resolution of an instances file location includes two file-system path components:

- The directory path to where the instances list file will be found:
  - default: `$DLC/servers/pasoe/conf`
- The instances list filename:
  - Default filename for UNIX: `instances.unix`
  - Default filename for Windows: `instances.windows`

The TCMAN administration utility looks for a private PAS for OpenEdge instances tracking configuration in one of two locations - in this order:

1. TCMAN looks for the `$DLC/servers/pasoe/conf/appserver.properties` file property: `psc.as.inst.dir`. This location works well when the directory path value is the name of an environment variable that each user can define a private location for. This location works less well for sharing an instance across multiple users by entering an absolute path.
2. If the `psc.as.inst.dir` property is not defined in the `$DLC/servers/pasoe/conf/appserver.properties` file, then TCMAN looks for the following OS process environment variable: `PSC_AS_INSTANCE_DIR`. This location produces the same results as using

the `psc.as.inst.dir` property, except that the `appserver.properties` file does not require tailoring (although the user's environment does).

3. If an instances list location is not yet configured after executing steps 1 and 2, then TCMAN uses the instances file in the default location (`$DLC/servers/pasoe/conf`). This works well when no special instance tracking or persistence is required.

PAS for OpenEdge product installation configuration

The PAS for OpenEdge product's installation configuration for Private PAS for OpenEdge Instance Tracking can be declared in one location and applied automatically for all users.

1. Define the property `psc.as.inst.dir` in `$DLC/servers/pasoe/conf/appserver.properties`.

   a. The value may be an absolute file path, where each user has their own private copy of that file-system path:

   ```
   psc.as.inst.dir=/wrkdir/data
   ```

   b. The value may be an OS process environment variable that will be defined in each user's shell startup and used filled in by the TCMAN utility:

   ```
   psc.as.inst.dir=${MYINSTDIR}
   ```

   In this case, each user is required to define the environment variable *MYINSTDIR*, with a user-chosen OS file path. For example:

   ```
   ##UNIX Shell initialization example:
   export MYINSTDIR=/wrkdir/data

   ##Windows Shell initialization example:
   MYINSTDIR | c:\wrkdir\data
   ```

User OS Process Environment Configuration

Each user may define specifically named OS process environment variables in their shell process startup script. The value of that environment variable provides the filesystem path to their private instances file.

UNIX shell initialization:

```
export PSC_AS_INSTANCE_DIR=os-dir-path
```

For example:

```
export PSC_AS_INSTANCE_DIR=/wrkdir/data
```

Windows user process environment configuration:

```
PSC_AS_INSTANCE_DIR os-dir-path
```

For example:

```
PSC_AS_INSTANCE_DIR  | c:\wrkdir\data
```

Delete a PAS for OpenEdge instance

System Administrators can delete an instance using PASMAN or OpenEdge Management.

Note: You cannot delete the PAS for OpenEdge installation directory (`$CATALINA_HOME`) using these procedures.

See also:

- Use PASMAN to delete an instance
- Use OpenEdge Management to delete an instance

Use PASMAN to delete an instance

Using the `pasman.[sh|bat] delete` command removes the directory tree and all of the files in an instance. You cannot recover any files or directories removed by the delete action. Backup anything you want to save before executing this action. Alias tracking is disabled for servers that are removed.

To delete a PAS for OpenEdge instance using PASMAN:

1. Run the Proenv utility.
2. Stop the target instance if it is running.
3. Type the `pasman.[sh|bat] delete` command along with the specified filepath and instance name:

   `pasman.[bat|sh] delete` *`instance-path/instance-name`*

Use OpenEdge Management to delete an instance

To delete a PAS for OpenEdge instance using OpenEdge Management:

1. On the **Resources** page, click the PAS for OpenEdge instance that you want to delete.

   The **Progress Application Server: *`instance name`*** page appears.
2. Stop the instance if it is running.
3. Select **Delete**. The instance is deleted.

3

Deploy an application in PAS for OpenEdge

As a system administrator, you should test any production configuration on a staging instance with production security before moving to a production environment in Progress Application Server for OpenEdge (PAS for OpenEdge).

Use the staging instance to:

- Configure the instance for production
- Stage r-code
- Deploy artifacts including ABL applications, ABL web applications, and ABL services
- Test and tune the instance

After testing and tuning are complete, you can export an ABL application as an OpenEdge Application Archive package and deploy it to a production instance, or create a ZIP file that includes both the r-code and the deployed artifacts so that the files can be extracted and registered on a production server. For more information, see OpenEdge Application Archive Structure on page 67 and Package an instance for production on page 88.

For details, see the following topics:

- Configure properties for deployment
- Stage the r-code
- OpenEdge Application Archive Structure
- Create an OpenEdge Application Archive using tcman export
- Create an OpenEdge Application Archive using an Ant Build
- Tailor an ABLApp installation using Ant

- Deployment artifacts
- ABL application PING service
- Package an instance for production

Configure properties for deployment

After you create a staging environment, you must configure values in the `openedge.properties` file of your instance and variables to suit your environment.

As a system administrator working in a staging environment, you:

- Set the PROPATH
- Set database connections
- Enable transports
- Set agents and sessions
- Modify environment variables (if required)

Set the PROPATH

Use the OEPROP utility to view or set the PROPATH, which is where the PAS for OpenEdge instance looks for r-code.

1. Run the Proenv utility.
2. Go to the `/bin` directory of the instance.
3. Run the OEPROP utility to display the PROPATH settings for the instance:

   `oeprop.[sh|bat] AppServer.Agent.`*`instance-name`*`.PROPATH`

4. Progress recommends that you add the following directories to the PROPATH:
   - *`instance-path/instance-name/`*`openedge`
   - *`instance-path/instance-name/`*`webapps/`*`ablwebapp-name/`*`WEB-INF/openedge`

Note:

- For application code, move the compiled code to *`instance-path/`*`webapps/ROOT/WEB-INF/openedge`.
- For common code used by multiple applications, move the compiled code to *`instance-path/`*`openedge`.

Set the database connections

Set the database connections for your instance in the staging environment. If you are running multiple ABL web applications from the instance, then set the individual database connections for each application.

1. Run the Proenv utility.
2. Go to the `/bin` directory of your instance.
3. Run the OEPROP utility to set the database connection for your instance:

```
oeprop.[sh|bat] AppServer.SessMgr.agentStartupParam="-db
    path-to-database"
```

4. Restart the instance using the `pasman pasoestart` command:

```
pasman.[sh|bat] pasoestart -I instance-name -timeout 200
```

Enable the transports

By default, all transports are disabled in the production version of PAS for OpenEdge. On a production instance, enable only those transports you plan to use.

To enable a transport, set the transport `adapterEnabled` property to `1` by editing the `openedge.properties` file with the OEPROP utility:

1. Run the Proenv utility.
2. Go to the `/bin` directory of your instance.
3. Run the OEPROP utility to display the transport settings for each of the transports:

```
oeprop.[sh|bat] instance-name.ablwebapp-name.APSV.adapterEnabled
oeprop.[sh|bat] instance-name.ablwebapp-name.REST.adapterEnabled
oeprop.[sh|bat] instance-name.ablwebapp-name.WEB.adapterEnabled
oeprop.[sh|bat] instance-name.ablwebapp-name.SOAP.adapterEnabled
```

4. Set the desired transports to `1`. All the other transports should be set to `0`:

```
oeprop.[sh|bat] instance-name.ablwebapp-name.transport.adapterEnabled=1
```

5. Restart the instance using the `pasman pasoestart` command:

```
pasman.[sh|bat] pasoestart -I instance-name -timeout 200
```

Set the agents and sessions

As a system administrator, you set a baseline number of agents and sessions available to the staging instance. As you test the instance, you may want to revisit these numbers and adjust them to improve performance.

To configure agents and sessions:

1. Run the Proenv utility.
2. Go to the `/bin` directory of your instance.
3. Run the OEPROP utility commands to view the agent and session settings for your instance:

```
oeprop.[sh|bat] AppServer.SessMgr.maxAgents
oeprop.[sh|bat] AppServer.SessMgr.maxConnectionsPerAgent
```

4. Set the number of agents and sessions:

```
oeprop.[sh|bat] AppServer.SessMgr.maxAgents=1
oeprop.[sh|bat] AppServer.SessMgr.maxConnectionsPerAgent=50
```

Note: The values you set for the instance will be different from the defaults and from those values listed in this topic. For information about tuning a production environment, see Tune PAS for OpenEdge instances.

5. Restart the instance using the `pasman pasoestart` command:

```
pasman.[sh|bat] pasoestart -I instance-name -timeout 200
```

6. Test and tune the instance, as necessary, for best performance. Tuning recommendations are covered in Tune PAS for OpenEdge instances.

Modify the environment variables

If you need to add additional environment variables that run each time an instance is launched, create a `*_setenv.sh` or `*_setenv.bat` file and add it to the *`instance-name`*`/bin` directory.

When the instance is started, all files with the suffix `_setenv.sh` or `_setenv.bat` run.

Note: Do not modify the existing `openedge_setenv.sh` or `openedge_setenv.bat` files.

Stage the r-code

For security reasons, you can only deploy compiled application code (known as run-time code, or r-code) to a staging instance of PAS for OpenEdge.

System administrators receive r-code from developers and place the code in the *`instance-name`*`/openedge` directory.

Note: If your ABL application code is monolithic, then you have the option to change the PROPATH using the OEPROP utility to reference the application externally. This is not recommended because it prevents you from creating a ZIP file with the complete instance and registering it on the production server.

OpenEdge Application Archive Structure

An OpenEdge Application Archive (OEAR) is specific type of zip file that contains web application resources that can be deployed to a PAS for OpenEdge instance in a single operation. It contains an Ant build script that gets executed during an import or export operation, and is where you can customize the configuration of your application.

An OpenEdge Application Archive is comprised of the following artifacts:

- `conf/`—The files in the `conf` directory will be in the target PAS for OpenEdge instance's `${CATALINA_BASE}/ablapps/${oepas-app}/conf` directory. This directory is required for generating and deploying an OpenEdge Application Archive.
  - `conf/MANIFEST.MF`—The `MANIFEST.MF` file contains a description of the ABL application and version name. See the "Manifest Attributes" section in this topic for more information.
  - `conf/oeablSecurity.properties`—The `oeablSecurity.properties` sets the security configuration for the ABL application of the target instance.
- `tlr/`—The files in the `tlr` directory are used for configuring the ABL application and will be in the the target instance's `${CATALINA_BASE}/ablapps/${oepas-app}/tlr` directory. This directory is required for generating and deploying an OpenEdge Application Archive.
  - `tlr/merge.properties`—The `merge.properties` file contains configuration settings relating to PROPATH, agent, and session manager settings. This file requires the ABL application name as a section header, and the webApps property be set to a comma separated list of web applications included in the ABL application. For example, if the ABL application name is SSOApp, the ABL application section looks similar to the following:

    ```
    [SSOApp]
        webApps=Payments,nginx,SSOProd
    ```

  - `tlr/build.xml`—The required `build.xml` file contains tailoring instructions for the ABL application. For more information about tailoring your `build.xml` file, see Tailor an ABLApp installation using Ant on page 73.
- `webapps/`—The applications in the `webapps` directory are deployed as part of the instance to the target location's `${CATALINA_BASE}/webapps/${oepas-webapp}` directory. The following web application types are supported:
  - `${oepas-webapp}.war`: self-contained archive for complete installs
  - `${oepas-webapp}.zip`: archive for overlays on a newly-deployed version of the current `${CATALINA_HOME}/extras/oeabl.war` .
- `openedge/`—The `openedge` directory contains r-code and other related artifacts for the ABL application to be deployed to the target instance's `${CATALINA_BASE}/ablapps/${oepas-app}/openedge` directory. This directory is recommended for generating and deploying an OpenEdge Application Archive.

An OpenEdge Application Archive can be comprised of these **optional** directories:

- `bin/`—The **optional** `bin` directory contains scripts relating to the ABL application to be deployed to the target `${CATALINA_BASE}/bin` directory.
- `instance/`—The **optional** `instance` directory contains additional directories that will be copied to `${CATALINA_BASE}`.

Note: The `tcman export` command currently does not support populating the `bin` and `instance` directories. In order to include these directories in your OpenEdge Application Archive, use the ABLApp Ant Task described in Create an OpenEdge Application Archive using an Ant Build on page 70.

Manifest Attributes

The manifest for the OpenEdge Application Archive specifies the following ABL application attributes:

Attribute Name	Sample Value	Description
`ABL-Application-Name`	`oepas1`	This value should match the folder name. If not, a warning is written into the deployment logs.
`ABL-Application-Version`	`1.0.0`	A semantic version identifier for this ABL application.

You can add additional attributes to the manifest as needed.

Create an OpenEdge Application Archive using tcman export

PAS for OpenEdge supports an application packaging strategy where an ABL application and its components are packaged as an **OpenEdge Application Archive (OEAR)**.

For more information about ABL applications, see OpenEdge Application Archive Structure on page 67 and "ABL applications" in *Learn about Progress Application Server (PAS) for OpenEdge*.

An ABL application can be packaged as an OpenEdge Application Archive (`.oear`) file, by running the `tcman export` command on the ABL application. For example:

```
oepas1/bin/tcman.sh export -appName oepas1
```

The `tcman export` command runs a series of Ant Tasks to produce an `oepas1.oear` file in the directory where the command is run.

The `tcman export` command includes a number of options which are described in "Export an ABL application (export)" in *PAS for OpenEdge Administration Tools Reference*.

One option to note is the `-full` property, which specifies a full export of oeabl.war-based web applications as WAR files and is shown in the following example:

```
oepas1/bin/tcman.sh export -appName oepas1 -full
```

If the `-full` option is present, all oeabl.war-based web applications are exported in full as WAR files (for example, ROOT.war). Otherwise, the `tcman export` script runs a diff-check on each oeabl.war-based web application against oeabl.war and only includes files modified or added into a zip file for each web application.

To learn about deploying an OpenEdge Archive file, see Deploy an OpenEdge Application Archive using tcman import on page 75.

Create an OpenEdge Application Archive using an Ant Build

If you use Ant or Gradle to compile your ABL code or for CI/CD purposes, you can use the ABLApp Ant Task to generate an OpenEdge Application Archive (OEAR) for an ABL application. The ABLApp Ant Task functionality is provided by `oeant-12.2.0.jar`. Using the ABLApp Ant Task lets you automate the creation of OpenEdge Application Archive packages using your source-control or other environments, which supports continuous integration and deployment practices.

For more information about ABL applications, see OpenEdge Application Archive Structure on page 67 and "ABL applications" in *Learn about Progress Application Server (PAS) for OpenEdge*.

To generate an OpenEdge Application Archive file (.oear) for an existing ABL application, include the following line in your `build.xml` file:

```
<taskdef name="ABLApp" classname="com.progress.oeant.ABLApp"
classpath="${env.DLC}/java/oe/oeant-12.2.0.jar"/>
```

The ABLApp Ant Task is a subclass of the Ant Zip task. The majority of the documentation for Ant Zip tasks can be found at https://ant.apache.org/manual/Tasks/zip.html

The ABLApp task also uses some code from the Jar task for handling Manifests. You can find documentation regarding manifests at https://ant.apache.org/manual/Tasks/jar.html.

Differences between ABLApp and Zip Ant Tasks

The following attribute from the Zip task is deprecated in the ABLApp task: `destFile`

It is replaced by the following attributes:

- `destDir`—The destination directory where the archive will go. The default is current directory.
- `archiveBaseName`—The basename that the archive will use. The default is the ABLApp name.
- `archiveClassifier`—Optional classifier that can be added to archive name (ALPHA, RELEASE, LATEST, etc). The default is blank.
- `archiveExtension`—Archive extension type. The default is `.oear`.
- `archiveVersion`—Optional version to use in the archive filename. The default is blank.

The archive filename is constructed as:

```
destDir/(archiveBaseName)-(archiveVersion)-(archiveClassifier)(archiveExtension)
```

Generating an ABLApp with Ant

Like the Ant Zip task, there are two ways to generate an OEAR file.

The first way uses `basedir` and requires you to copy over all the resources required by your ABLApp and ensure they are in the proper location.

The second way is to use filesets to pull resources from different locations an assemble them in the archive.

In either case, `basedir` must be set. It should point to an empty directory if you are using filesets.

Embedded attributes

The following attributes can be used as nested elements and to specify filesets. See the Jar task documentation for more information.

- `<tlr>`—Fileset that comprises resources in the `$CATALINA_BASE/ablapps/{ABLAppName}/tlr` directory
- `<webapps>`—Fileset that comprise the set of web applications used by this ABL Application. It must be a fully tailored war file, or a zip file that will be overlaid over a tailored `oeabl.war`.
- `<openedge>`—Fileset that comprises the `$CATALINA_BASE/ablapps/{ABLAppName}/openedge` directory.
- `<conf>`—Fileset that comprises the `$CATALINA_BASE/ablapps/{ABLAppName}/conf` directory

The following attributes require an update to the import script for them to be installed with the Import command:

- `<bin>`—Fileset that comprises resources in the `$CATALINA_BASE/bin` directory.
- `<instance>`—Fileset for instance-level resources.

The above can be used in conjunction with `basedir`, but should not be used with `instancedir`.

Additional attributes

The following additional attribute is supported in an ABLApp Ant Task:

- `instancedir`—Location of a PAS for OpenEdge instance that you are exporting an ABLApp from.

  You can not specify both `basedir` and `instancedir` in your Ant script.

Example Ant Script

The following example Ant script uses the ABLApp task to generate an OpenEdge Application Archive (OEAR) package. In this example, the OEAR package loads the Progress Compiler Tools Jar file, creates an OpenEdge database, contains three web applications (nginx, Payments, and SSOProd), and compiles ABL code:

```
<project name="OpenEdge Application Archive (OEAR) Example" default="build" basedir=".">


    <description>
    Generates an OpenEdge Archive file that can be deployed to a PASOE Server
    </description>

    <!-- Set environment variables required for the ANT script -->
    <property environment="env" />
    <fail unless="env.DLC" message="You should install OpenEdge to run this script"/>
    <property name="openedge" value="src/main/openedge" />
    <property name="bin" value="src/main/bin" />
    <property name="tlr" value="src/main/tlr" />
    <property name="conf" value="src/main/conf" />
    <property name="rcode" value="build/rcode" />
    <property name="instance" value="src/main/instance" />
    <property name="dbdir" value="build/db" />
    <property name="webapps" value="build/webapps" />
    <property name="theAppName" value="SSOApp" />
    <property name="theAppVersion" value="1.0.0" />
 <property name="classifier" value="ALPHA" />


    <!-- Load PCT tasks from DLC/pct -->
   <taskdef resource="PCT.properties" classpath="${env.DLC}/pct/PCT.jar" loaderRef="pct"
 />
    <typedef resource="types.properties" classpath="${env.DLC}/pct/PCT.jar"
loaderRef="pct" />
```

```
    <record name="build.log" loglevel="verbose" append="false"/>

    <target name="build" depends="clean,packageApp">
    </target>

    <target name="clean" >
        <delete quiet="true" >
         <fileset dir="build" />
     </delete>
    </target>

    <target name="packageApp" depends="compile, buildwebapps">
        <taskdef name="ABLApp" classname="com.progress.oeant.ABLApp"
classpath="${env.DLC}/java/oe/oeant-12.2.0.jar"/>


        <ABLApp description="Sample creation of an ABLApp"
            appName="${theAppName}"
            archiveBaseName="${theAppName}"
            archiveVersion="${theAppVersion}"
            archiveClassifier="ALPHA"
            basedir="src/main/SSOApp"
            destdir="build/oear">

            <bin dir="${bin}" />
            <tlr dir="${tlr}" />
            <conf dir="${conf}" />
            <openedge dir="${rcode}" includes="**/*.r,*.gen,*.map" />
            <webapps dir="${webapps}" />
            <instance dir="${instance}" />

        </ABLApp>

    </target>

    <target name="compile" depends="createdb">

        <PCTCompile destDir="${rcode}" dlcHome="${env.DLC}" relativePaths="false">
            <fileset dir="${openedge}" includes="**/*.p,**/*.cls" />
            <DBConnection dbName="sports2000" dbDir="${dbdir}" readOnly="true" />
            <propath>
                <pathelement path="${openedge}"/>
            </propath>
        </PCTCompile>

        <copy todir="${rcode}">
            <fileset dir="${openedge}">
                <include name="**/*.gen,**/*.map"/>
            </fileset>
        </copy>

    </target>

    <target name="createdb">
     <echo>creating database copy of sports2000</echo>

     <mkdir dir="${dbdir}"/>
     <sports2000 dbName="sports2000" destDir="${dbdir}" dlcHome="${env.DLC}" />

    <PCTLoadData dlcHome="${env.DLC}" srcFile="${openedge}/_sec-authentication-domain.d"
 table="_sec-authentication-domain">
         <DBConnection dbName="sports2000" dbDir="${dbdir}" singleUser="true"/>
     </PCTLoadData>

    </target>

    <target name="buildwebapps">
        <echo>building web application zip files</echo>
```

```
        <zip destfile="build/webapps/nginx.zip" basedir="src/main/webapps/nginx" />
        <zip destfile="build/webapps/Payments.zip" basedir="src/main/webapps/Payments"
 />
        <zip destfile="build/webapps/SSOProd.zip" basedir="src/main/webapps/SSOProd"
/>

    </target>

</project>
```

The output of the Ant script is SSOApp-1.0.0-ALPHA.oear.

Tailor an ABLApp installation using Ant

The default build.xml file in the *instance-name*/ablapps/*app-name*/tlr directory serves as a template build file for ABL application deployment tailoring. This can be customized to fit your build needs during application deployment, post-application deployment, and during other application lifecycle events. The following targets are defined and called by our tooling.

- ablapp_deploy—Called when an ABLApp is being prepared to be imported.
- ablapp_deployed—Called when an ABLApp is finished being imported.
- ablapp_export—Called when an ABLApp is being prepared to be exported.
- ablapp_exported—Called when an ABLApp is finished being exported.

PAS for OpenEdge tooling currently does not have support for undeploying an ABL application in a single command. However, undeploying a web application that is part of the ABLApp will cause these targets to be called. This is done multiple times as each web application that comprises the ABLApp is undeployed:

- ablapp_undeploy—Called when an ABLApp is being prepared to be undeployed.
- ablapp_undeployed—Called when an ABLApp is being prepared to be imported.

The following tailoring build.xml file provides an example of how to set environment variables, compile ABL code, and create and start an OpenEdge database when the ABL application is deployed to a PAS for OpenEdge instance:

```
<project name="PASOE SSO with form-local OpenEdge archive" basedir=".">

  <description>
  Template build file for ABL application deployment tailoring.
  Added to instance ablapps appname tlr directory.
  </description>

  <!-- Set environment variables required for the ANT script -->
  <property environment="env" />
  <fail unless="env.DLC" message="You should install OpenEdge to run this script"/>
  <property name="openedge" value="../openedge" />
  <property name="dbdir" value="${catalina.base}/work" />
  <property name="rcode" value="../openedge" />
  <property name="ablappconf" value="${catalina.base}/ablapps/${ablapp.name}/conf" />

  <condition property="isWindows">
    <os family="windows" />
  </condition>
```

```
  <condition property="isUnix">
    <os family="unix" />
  </condition>

  <!-- Called when a deployment artifact is created -->
  <target name="ablapp_deploy">
   <echo>creating database copy of sports2000</echo>

   <mkdir dir="${dbdir}"/>
    <exec executable="${env.DLC}/bin/prodb">
      <arg value="${dbdir}/sports2000"/>
      <arg value="sports2000"/>
    </exec>
  </target>

  <target name="ablapp_deployed">
    <!-- Called when a deployment artifact is created -->

    <!-- assume ROOT is not expanded -->
    <unzip src="${catalina.base}/webapps/ROOT.war"
dest="${catalina.base}/webapps/ROOT/"/>

    <!-- Copy over redirect page to ROOT-->
    <copy file="errorPage.jsp" todir="${catalina.base}/webapps/ROOT/WEB-INF/jsp/"
overwrite="true"/>

  </target>

  <target name="ablapp_undeploy">
    <!-- Called when a deployment artifact is removed from its container -->
  </target>

  <target name="ablapp_undeployed">
    <!-- Called when a deployment artifact is removed from its container -->
  </target>

  <target name="ablapp_export">
    <!-- Called when a deployment artifact is exported from its container -->
  </target>

  <target name="ablapp_exported">
    <!-- Called when a deployment artifact is exported from its container -->
  </target>

</project>
```

Deployment artifacts

Each PAS for OpenEdge instance configuration contains a default web application called ROOT. The ROOT application is a starting point for deploying new services. For simple services and early development, you can deploy artifacts to ROOT and then create a ZIP file of the entire instance to easily share it with others, or create an OpenEdge Application Archive of the ABL application which contains the ROOT web application. For some applications, this may be all that is required. As your services evolve, you can create your own web applications that group related services into separate deployment packages for security or user permissions reasons.

Deploy services to the existing web application, ROOT, or generate a PAS for OpenEdge web application from Progress Developer Studio that is deployed outside of ROOT. If you want to replace ROOT with another web application, then undeploy ROOT, and deploy the web application that does what you want when a client makes a request to your server with no path specified. This can be a login page, a redirect, or something else, depending on your needs.

This topic focuses on deploying services to the ABL web application, ROOT.

You can deploy the following artifacts to a PAS for OpenEdge instance:

- Individual SOAP service descriptors (`.wsm` files) to an existing ABL web application.
- Individual REST service definitions (`.paar` files) to an existing ABL web application.
- ABL and WebHandler services to an existing ABL web application.
- One or more new ABL web applications and their services as a `.war` file.
- An ABL application as an OpenEdge Application Archive, which contains ABL web applications and services.

System administrators can deploy an ABL application as one or more ABL web applications:

- Deploy a single ABL web application by packaging it in one `.war` file that follows the standard web application directory structure. For improved performance and resource consumption, deploy an ABL application using an ABL web application—making the full application accessible from a single server URL.
- There are situations where deploying an ABL application as multiple ABL web applications (where the application is accessible across multiple server URLs) is desirable. Because using multiple ABL web applications slows performance and consumes more OS process resources, use this approach only when your ABL web application is designed as a set of discrete, individually deployed, services where the ABL web application encapsulates a "service set" within your ABL application.

The standard directory structure for a web application is:

- *`instance-name`*`/webapps/`*`webapp-name`*—This directory is the root directory for the application by default. The root directory contains the `/WEB-INF` folder, and any files that are visible to the client browser, like HTML and JSP pages. These files can be organized into a subdirectory for more complicated applications.
- *`instance-name`*`/webapps/`*`webapp-name`*`/WEB-INF`—This folder contains the web application deployment descriptor (`web.xml`) that contains the configuration information for the application in a standard format.

When deploying ABL web applications, you can pre-deploy and configure transports. This content focuses on deploying services to the existing ROOT application. You also have the option of deploying the SOAP, REST, and APSV service interface API `.r` files as part of each web application `WEB-INF/openedge` directory. If you optionally embed `.r` modules in the web application, then you ensure that the version of the transport interfaces remains consistent with their transport (URI) definitions. You may need to change the PROPATH of the ABL application to include `WEB-INF/openedge` path of the ABL web application.

See also:

- Deploy an ABL application as an OpenEdge Application Archive (.oear)
- Deploy ABL services
- Deploy REST services
- Deploy SOAP services
- Deploy WebHandler services
- Deploy an ABL web application
- Deploy remote administration applications

Deploy an OpenEdge Application Archive using tcman import

PAS for OpenEdge supports an application deployment strategy where an ABL application and its components are deployed as an **OpenEdge Application Archive (OEAR)**.

For more information about ABL applications, see "ABL applications" in *Learn about Progress Application Server (PAS) for OpenEdge*.

To deploy an ABL application as an OpenEdge Application Archive (`.oear`) file, follow these steps:

1. Obtain the OpenEdge Application Archive (`.oear`) file. Information about creating OpenEdge Application Archives can be found in Create an OpenEdge Application Archive using tcman export on page 69 and Create an OpenEdge Application Archive using an Ant Build on page 70.
2. On the target instance where you want to deploy the OEAR, undeploy the ROOT web application, and any web application that exists in the OEAR package. For example:

```
proenv>importInstance\bin\tcman.sh undeploy ROOT
OK - undeployed application ROOT from local directory
C:\OpenEdge\WRK122\importInstance\webapps
```

3. From the `/bin` directory of your target instance, run the `tcman import` command with the path to the OpenEdge Archive (`.oear`) file. For example:

```
proenv>importInstance\bin\tcman import .\oepas1.oear
Importing archive C:\OpenEdge\WRK122\oepas1.oear
Buildfile: C:\OpenEdge\WRK122\importInstance\temp\appimport\tlr\build.xml

ablapp_deploy:

BUILD SUCCESSFUL
Total time: 0 seconds
OK - deployed C:\Progress\OPENED~2\servers\pasoe\extras\oeabl.war to local directory
 C:\OpenEdge\WRK122\importInstance2\webapps as application ROOT
Buildfile: C:\OpenEdge\WRK122\importInstance\ablapps\oepas1\tlr\build.xml

ablapp_deployed:

BUILD SUCCESSFUL
Total time: 0 seconds
Successfully imported oepas1 to C:\OpenEdge\WRK122\importInstance
```

   The `tcman import` command adds the ABL application to the `openedge.properties` file of the instance, adds any security policies for the ABL application to the instance, deploys all web applications to the instance, and adds their details to `openedge.properties` file.

   During the import into the target instance, the `tcman import` command reads the `build.xml` script for tailoring instructions and outputs them to the console. To learn more about tailoring a `build.xml` file, see Tailor an ABLApp installation using Ant on page 73.

A few things to note when deploying an OEAR using the `tcman import` command:

- If you are deploying an OEAR into an instance with existing ABL applications or ABL web applications with the same names as those included in the OEAR package, you must undeploy those applications using the `tcman undeploy` command before importing the new OEAR file and applications. Undeploying all web applications from an ABL application removes the ABL application from the instance.

  For more information, see "tcman undeploy" in *PAS for OpenEdge Administration Tools Reference.*

- If your OEAR package includes a ROOT web application, you must undeploy ROOT on the target instance.
- The `tcman import` command does not patch or update applications on the target instance. It replaces existing applications.

For more information about `tcman import` syntax and usage, see "Import an ABL application (import)" in *PAS for OpenEdge Administration Tools Reference*.

Deploy REST services

To deploy a REST service, deploy the `.paar` file to an existing web application like ROOT, or whichever ABL web application requires the service.

The deployREST command-line tool adds REST services to a deployed ABL web application (`oeabl.war`). After you deploy the service, stop and restart the instance, to run the new service.

To deploy REST services to an ABL web application for an existing instance:

1. Get the `.paar` file and any ABL r-code that supports the API for the service interface from the developer.
2. Move the r-code to the PROPATH.
3. Go to the `/bin` directory of the instance.
4. Use the deployREST command to deploy the `.paar` file to the ABL web application:

```
deployREST.{bat|sh} path_to_rest/RESTservice.paar ABLwebapp
```

To undeploy a REST service, run the deployREST command with the `-undeploy` option specified, for example:

```
deployREST.{bat|sh} path_to_rest/RESTservice.paar ABLwebapp -undeploy
```

Deploy ABL services

You can run ABL services in an existing ABL web application. There is no deploy command for ABL services. The only requirement is that the r-code is defined in the PROPATH of the instance

To deploy an ABL service using the APSV transport:

1. Ensure that the APSV transport is enabled.
2. Move the r-code to the PROPATH.

Deploy SOAP services

The deploySOAP command-line tool adds SOAP services to a deployed ABL web application (`oeabl.war`). After you deploy the SOAP service, stop and restart the PAS for OpenEdge instance to run the new service.

To deploy SOAP services to an ABL web application for an existing instance:

1. Get the `.wsm` file and any ABL r-code that supports the API for the service interface from the developer.
2. Move the r-code to the PROPATH.
3. Go to the `/bin` directory of the instance.
4. Use the deploySOAP command to deploy the `.wsm` file to your ABL web application:

```
deploySOAP.[bat|sh] path_to_soap/SOAPservice.wsm ABLwebapp
```

Deploy web handler services

Web handler deployment has changed for OpenEdge 12.2, where web handlers are no longer specified in `openedge.properties`, but through an information file for each web service called either `ROOT.handlers` or *`webapp-name`*`.handlers`. When you deploy a web handler service to a PAS for OpenEdge production instance, the WEB transport must be enabled.

Deploy new web handlers in OpenEdge 12.2

New web handlers in OpenEdge 12.2 are deployed using separate information files for each ABL web application and service deployed to an instance. These information files should be placed in the *`instance-name`*`/webapps/`*`webapp-name`*`/WEB-INF/adapters/web/`*`svcName`* directory.

Perform the following steps to deploy a new web handler to an instance:

1. Get the r-code that supports the web handler from the service developer as a zip file.
2. Extract the r-code supporting the web handler and move it to the PROPATH. For example, *`instance-name`*`/webapps/`*`webapp-name`*`/WEB-INF/openedge`
3. Open the proenv utility, and then navigate to the `/bin` directory of the target instance where you want to deploy the web handler.
4. Run the `deploySvc` command and specify the web application name and the location of the zip file. For example, to deploy the `myWebSvcApp` service to the ROOT web application, enter this command:

   ```
   deploysvc.{sh|bat} -a ROOT myWebSvcApp.zip
   ```

   To deploy the web service to a different web application, specify the web application name with the `-a` option.
5. If you want to deploy the web handler service to the ROOT web application, create a `ROOT.handlers` file, which describes the composition of the web handler and place it in *`instance-name`*`/webapps/ROOT/WEB-INF/adapters/web/ROOT` directory.

   Now, create the above directory structure manually and place the web handler file. The web handler information file (`.handlers`) uses a JSON format to hold the version number, service name, and details of the ABL web handler.

   Example:

   ```
   {
   "version": "2.0",
   "serviceName": "",
   "handlers": [
   {"uri":"/myhandler","class":"MyWebHandlerClass"}
   ]
   }
   ```
6. Start the PAS for OpenEdge instance.
7. Test the web handler in a web browser. If you deployed a web handler to the ROOT web application, the URL can be accessed using the following format:

   ```
   http://hostname:port/web/svcName/URI
   ```

   Note: The URI is specified in the `ROOT.handlers` file.

If you deployed a web handler to another web application (for example, myPas) the URL can be accessed using the following format:

```
http://hostname:port/myPas/web/svcName/URI
```

Migrate web handlers to OpenEdge 12.2

System administrators can use the tcman export command to migrate web handlers from the old format of the openedge.properties file into the new web handlers JSON format Version 2.0.

When migrating web handler configurations from openedge.properties to the web handlers JSON format, the migrated web handlers will be added to the ROOT.handlers file, and the URLs will have backward compatibility with the older version.

Perform the following steps to migrate openedge.properties web handlers to the web handlers JSON format supported in OpenEdge 12.2:

1. Obtain the r-code that supports the web handler service for the OpenEdge 12.1 or prior instance from the developer.
2. Create a target PAS for OpenEdge instance using OpenEdge 12.2. If you are not deploying to the ROOT ABL web application, deploy a supporting ABL web application to this instance.
3. Move the r-code to the PROPATH of the OpenEdge 12.2 instance. For example, *instance-name*/webapps/ROOT/WEB-INF/openedge or *instance-name*/webapps/*webapp-name*/WEB-INF/openedge.
4. From the /bin directory of the target OpenEdge 12.2 instance, run the tcman export command on the older instance to export the ABL application and all of its supporting web applications and services into an OpenEdge Application Archive file. For example:

   ```
   proenv>WRK122\oepas122\bin\tcman export -appName oepas1 -instanceDir
   C:\OpenEdge\WRK121\oepas1\
   ```

   This process reads the web handler configuration from openedge.properties, strips them from the file, and adds them to the WEB-INF/adapters/web/ROOT/ROOT.handlers file.

   For example, if you had an ABL application called "myABLApp" with the web applications "SportsWebApp" and "EntityWebApp", and each web application has their web handler configuration in the openedge.properties file, then the exported OpenEdge Application Archive will contain the following handler files:

   ```
   <CATALINA_BASE>/webapps/SportsWebApp/WEB-INF/adapters/web/ROOT/ROOT.handlers
   <CATALINA_BASE>/webapps/EntityWebApp/WEB-INF/adapters/web/ROOT/ROOT.handlers
   ```

5. Deploy this OpenEdge Application Archive package to the OpenEdge 12.2 instance using tcman import. For example:

   ```
   proenv>WRK122\oepas122\bin\tcman import C:\OpenEdge\oepas1.oear
   ```

6. Start the OpenEdge 12.2 instance.
7. Test the web handler service in a web browser. It can be accessed using the same URI as the previous version of the web handler service.

Deploy web handlers prior to OpenEdge 12.2

When deploying a new web handler service in OpenEdge 12.0 and 12.1, add the custom handlers that are defined by the developer, in the order provided, to the `openedge.properties` file.

Perform the following steps to deploy a web handler service to an instance:

1. Get the r-code that supports the web handler service, and the defined `handler` names in the `openedge.properties` file, from the developer.
2. Move the r-code to the PROPATH.
3. Run the Proenv utility and go to the `/bin` directory of the target instance.
4. Add the `handler` properties to the `openedge.properties` file using the OEPROP utility:

   `oeprop +`*`instance-name.webapp-name`*`.WEB.handler1=`*`CustomHandler`*

Reload web handlers online

System administrators can reload web handler contexts without restarting the PAS for OpenEdge instance so that web handlers can be updated or published dynamically online. Administrators can use the `tcman jmx refreshWeb` command to reload the web handler context whenever a web handler is published or updated.

For more information, see "Refresh web handlers for an ABL web application (jmx refresh) in *PAS for OpenEdge Administration Tools Reference*.

Supported JSON formats for web handler information file

Two versions of the web handlers file are supported in OpenEdge 12.2, but Version 2.0 of the web handlers file can expect long-term support.

Figure 1: Web handlers JSON Version 1.0

```
{
    "version": "1.0",
    "serviceName": "SportsSvc",
    "handlers": [{
        "className": "SportsSvc",
        "URIs": [
            "\/webHandlers",
            "\/some-data",
            "\/customers\/{custnum}\/orders\/{ordernum}",
            "\/customers\/{custnum}\/orders",
            "\/customers\/{custnum}",
            "\/customers"
        ]
    }]
}
```

The web handlers JSON Version 1.0 is the format that is generated by Progress Developer Studio in OpenEdge 12.2.

Figure 2: Web handlers JSON Version 2.0

```
{
  "version": "2.0",
  "serviceName": "",
  "handlers": [
    {
      "uri": "/WebHandlers",
      "class": "WebHandlersHandler",
      "enabled": true
```

```
    },
    {
      "uri": "/customers/{custnum}/orders/{ordernum}",
      "class": "CustomerService",
      "enabled": true
    }
  ]
}
```

Web handlers JSON Version 2.0 is the format generated when migrating older web handler properties using `tcman export`.

Deploy an ABL web application

System administrators can deploy ABL web applications to a PAS for OpenEdge instance using the `pasman deploy` command.

To deploy an ABL web application to a PAS for OpenEdge instance, perform the following steps

1. Get the `.war` file and any ABL r-code that support the ABL web application from the developer.
2. Move any r-code to the PROPATH, for example:

   ```
   instance-name/openedge
   ```

3. Deploy the ABL web application to the instance using the `pasman` utility:

   ```
   pasman.[sh|bat] deploy -I instance-name path_to_ABLwebapp/ABLwebapp.war
   ```

4. Start the instance:

   ```
   pasman.[bat|sh] pasoestart -I instance-name -timeout 200
   ```

5. Verify that the ABL web application is deployed by calling it in a web browser:

   ```
   http://hostname:port/ABLwebapp
   ```

   - *`hostname`* is the instance name.
   - *`port`* is the port defined for the instance.
   - *`ABLwebapp`* is the name of the war file that you deployed.

You can have more than one ABL application running on an instance depending upon your business requirements. Each ABL web application should have its own database connection and security.

To create an additional ABL web application for an instance, deploy the `oeabl.war` file located in the PAS for OpenEdge core installation:

```
pasman.[bat|sh] deploy -I instance-name –a webapp-name
$DLC\servers\pasoe\extras\oeabl.war
```

Note: Additional ABL web applications must have a different name than existing ABL web applications in the PAS for OpenEdge instance.

The *webapp-name* is then added in the `[Appserver]` section in `openedge.properties` file. Each deployed ABL web application can be configured individually using the OEPROP utility in the `AppServer.Sessmgr.`*webapp-name* section. Using OEPROP to set these properties is described in Configure properties for deployment.

To enable the secure online deployment of new ABL applications, see Secure online deployment of a new ABL application.

Deploy remote administration applications

Note: This topic is optional. Remote administration applications help monitor the performance of the instance, but they can pose security risks if they are not properly configured.

You have a number of options for monitoring and managing a PAS for OpenEdge instance in a production environment:

- Log in to the production instance and use PASMAN locally.
- Deploy remote administration applications such as Tomcat Manager and OpenEdge Manager to the PAS for OpenEdge instance.

If you choose to use remote management tools, you need to manually deploy them to the production instance. You must also ensure that you have secured your environment against unauthorized access. For more information, see Secure online deployment of a new ABL application.

Run this command in the `bin` directory of the instance to deploy the Tomcat manager application:

```
pasman.[sh|bat] deploy -I instance-name $DLC\servers\pasoe\extras\manager.war
```

Run this command in the `bin` directory of the instance to deploy the OpenEdge manager application:

```
pasman.[sh|bat] deploy -I instance-name $DLC\servers\pasoe\extras\oemanager.war
```

ABL Service Deployment

An ABL service is a set of business logic that can be accessed through a URL path. The ABL Service Deployment feature allows users to package their ABL services into one deployment action, which allows users to build a custom service quickly and to easily deploy to a PAS for OpenEdge instance. This creates a single archive file (`.oeds`) for each transport, which makes the ABL Service easy to maintain and benefit from CI/CD processes.

OpenEdge Data Service (.oeds) file format

The `.oeds` file is a zip file with a predefined structure, it represents a single ABL service:

Location	Description	Contents	Target location	Applicable transport
`./conf`	The configuration directory. This is required.	`.MANIFEST.MF`	None	`APSV, REST, SOAP, STATIC, WEB`
`./openedge`	The r-code and other related artifacts for the ABL application. This is recommended.	`**/*.r **/*.cls **/*.p **/*.w **/**`	[illegible]	`APSV, REST, SOAP, WEB`

`./static`	Static files to be copied into the ABL webapp's static folder. These are files that can be accessed by any client. They are not private to the application.
	This is optional.
		Various files, including:
		`**/*.js`
		`**/*.json`
		`**/*.xml`
		`**/*.css`
		`**/*.png **/*.*`
			[illegible]	`REST, STATIC, WEB`

`./tlr`	Files in this folder are merged with those in the target location.
	Recommended: `oeablSecurity.merge`: The security properties are merged into the existing security intercept URL properties
	Optional: `build.xml`: contains tailoring for this abl-app:
	• `ABLSVC_DEPLOY, ABLSVC_DEPLOYED`
	• `ABLSVC_UNDEPLOY, ABLSVC_UNDEPLOYED`
	• `ABLSVC_PATCH, ABLSVC_PATCHED`
		`build.xml`
		`merge.properties`
		`merge.oeablSecurity`
			n/a
			[illegible]
			[illegible]
				`APSV, REST, SOAP, STATIC, WEB`

Location	Description	Contents	Target location	Applicable transport
./svc	For REST and SOAP, the artifacts are deployed via the deployREST and deploySOAP tools. For the MAP file, a simple copy into the webapp's WEB-INF/openedge folder. The HANDLERS file contains the configuration of the webhandler(s) and the URI's they handle. The ${oepas-ablsvc}.json file is the Data Object Service catalog, and is copied to the ./static folder. This is recommended.	${oepas-ablsvc}.paar ${oepas-ablsvc}.wsm ${oepas-ablsvc}.map/.gen ${oepas-ablsvc}.handlers ${oepas-ablsvc}.json	${CATALINA_BASE}/webapps/${oepas-webapp}/WEB-INF/adapters/rest ${CATALINA_BASE}/webapps/${oepas-webapp}/WEB-INF/adapters/soap ${CATALINA_BASE}/webapps/${oepas-webapp}/WEB-INF/openedge ${CATALINA_BASE}/webapps/${oepas-webapp}/WEB-INF/adapters/web ${CATALINA_BASE}/webapps/${oepas-webapp}/static	REST, SOAP, WEB
./custom	There may be additional folders in the archive; application administrators will need to deploy these artifacts into the appropriate directory by tailoring the build.xml script. This is optional.	*.*	A custom location depending on your setup and tailoring.	APSV, REST, SOAP, STATIC, WEB

ABL-ablsvc MANIFEST.MF attributes

The ABL application contains a manifest file that is a description of the ABL application which includes version information and attributes in the webapp's MANIFEST.MF defining the tailoring task names.

In addition to the common attributes, the following attributes are supported:

Attribute name	Sample value	Description
ABL-Service-Name	CustOrdSvc	This value should match the folder name. If not, a warning must be written into the deployment logs
ABL-Service-Version	0.0.0.0	The version of the service should match the version number in the filename.
ABL-Service-Type	REST,SOAP,WEB,STATIC,APSV	The type of service.

ABL application PING service

The PING service allows an ABL client to find out the state of a data service (online or offline) and the ABL application's ability to respond. This service can be used in conjunction with other static service resources (such as `/static/home.html`), and the HTTP status to determine whether the server, service, transport, and ABL application are available for use.

The APSV, REST, and WEB transports use the `ServerStatus()` method of the `OpenEdge.Rest.Admin.AppServerStatus` class to implement the PING service. You can extend the class to return application-specific information about database connections, PROPATH, loaded libraries, caches, available services, and more.

Note: The PING service is not available for the SOAP and Web transport.

Transport Syntax

REST

```
GET { http | https }://host:port/web_app/rest/_oepingService/_oeping
```

WEB

```
GET { http | https }://host:port/web_app/web/_oepingService/_oeping
```

To enable the PING service for the WEB transport, you must configure a handler in the `openedge.properties` file:

```
handlern=class name : /uri
```

Note: When a GET method is invoked on `/_oepingService/_oeping`, the `HandleGet` method in the class that is configured in the web handler is invoked. The response body in `HandleGet` is displayed on the web page as output.

The PING service only responds to the URL as it is configured in the `openedge.properties` file.

APSV

```
Run OpenEdge/ApplicationServer/Util/apsv_oeping.p on happsrv(OUTPUT
    JSON_Data).
```

happsrv is the application server handle, and *JSON_Data* is the JSON output.

JSON output format

When the PING service is successful, it can return a JSON array with the following information:

```
{“response”: {“_retVal”:  “text” }}
```

Note:

If nothing is returned to the PING service, then that indicates that the agent is available, since the default value for *text* is blank (" ").

To customize the return value, extract the `OpenEdge.Rest.Admin.AppServerStatus` class from the *oe-install-dir*`/src/OpenEdge.ServerAdmin.pl` procedure library, and place the customized version in your PROPATH.

If the PING service is not successful, then it returns a JSON array with the following information:

```
{"_errors": [ {"_errorMsg": "error_text",
        "_errorNum": error_number} ]}
```

Note that the error message may include additional properties.

Secure the PING service

You can restrict access to the PING service in the */web-app/*`WEB-INF/oeablSecurity.csv` file. For example, to restrict everyone from accessing REST PING, add the line in bold:

```
############## Intercept-url definitions for the REST transport URIs###############
"/rest/_oepingService/_oeping",”*”,"denyAll()"
"/rest/**","*","hasAnyRole('ROLE_PSCAdmin')"
    ###################################
```

Note: Order is important. The line with `denyAll()`, which restricts access to the PING service, must appear before the line that follows, which allows access to all REST services.

Example

The following code returns the server status that includes the names of connected databases and PROPATH entries.

```
method public character ServerStatus( ):
        define variable statusJson as JsonObject no-undo.
        define variable dataArray as JsonArray no-undo.
        define variable svrStatus as character no-undo.
        define variable cnt as integer no-undo.
        define variable maxCnt as integer no-undo.

        assign statusJson = new JsonObject().
          dataArray  = new JsonArray().
        statusJson:Add('ldb',dataArray).

        do cnt = 1 to num-dbs:
          dataArray:Add(ldbname(cnt)).
        end.

        assign maxCnt  = num-entries(propath)
          dataArray  = new JsonArray().
        statusJson:Add('propath', dataArray).
        do cnt = 1 to maxCnt:
          dataArray:Add(entry(cnt, propath)).
        end.

        statusJson:Write(output svrStatus).

        return svrStatus.
        end.
```

The output from the example code looks similar to the following:

```
{
        "response": {
              "_retVal": {
                    "ldb": [
                           "sports2020"
                           ],
                           "propath": [
        "c:\\username\\pas\\117\\baseinstall\\webapps\\ROOT\\WEB-INF\\openedge",

        "c:\\username\\pas\\117\\baseinstall\\openedge",

        "c:\\username\\dlc\\oe117\\tty",
        "c:\\username\\dlc\\oe117\\tty\\ablunit.pl",
        "c:\\username\\dlc\\oe117\\tty\\adecomm.pl",
        "c:\\username\\dlc\\oe117\\tty\\adecomp.pl",
        "c:\\username\\dlc\\oe117\\tty\\adeedit.pl",
        "c:\\username\\dlc\\oe117\\tty\\adeshar.pl",
        "c:\\username\\dlc\\oe117\\tty\\dataadmin.pl",
        "c:\\username\\dlc\\oe117\\tty\\OpenEdge.BusinessLogic.pl",

        "c:\\username\\dlc\\oe117\\tty\\OpenEdge.Core.pl",
        "c:\\username\\dlc\\oe117\\tty\\OpenEdge.ServerAdmin.pl",

        "c:\\username\\dlc\\oe117\\tty\\prodict.pl",
        "c:\\username\\dlc\\oe117",
        "c:\\username\\dlc\\oe117\\bin"
        ]
      }
    }
}
```

Package an instance for production

When the application artifacts are staged in the PAS for OpenEdge instance, you can package the instance as a ZIP file and extract it to the production instance.

To register a copy of a PAS for OpenEdge instance on a production server:

1. After testing and tuning an instance on a staging environment, create a ZIP file of the instance.
2. Copy and extract the instance into the new production server.
3. Register the instance by running the PASMAN register command:

   ```
   pasman.[sh|bat] register instance-name instance-path
   ```

   The register action enables tracking and also updates the value of `CATALINA_HOME` in all of the executable scripts in the `bin` directory of the instance so it refers to the new production server.

4

Secure PAS for OpenEdge instances

As a system administrator, you should consider doing the following before running the production version of PAS for OpenEdge:

- Add a valid key/certificate (SSL/TLS) to your PAS for OpenEdge instance. Do not rely on the private key/digital certificate sent by OpenEdge for testing in production. For more information, see Use TLS in PAS for OpenEdge on page 124.
- For server administration, add new users and assign PAS roles (`ROLE_PSC**`).
- Modify the Tomcat Manager users in `~/conf/tomcat-users.xml` if you add the Tomcat Manager application.
- Modify the PAS for OpenEdge Manager users `~/conf/tomcat-users.xml` if you add the OpenEdge Manager application. Remove the default username and password.
- Provide file system access as described in Manage users and remote access.
- Use remote access filters to control client IP addresses.
- Enabling remote administration and monitoring (if required) as described in Secure online deployment of a new ABL application.
- Understand and implement an ABL application client login model and user account source as described in Enable ABL application authentication.
- Learn how to turn off HTTP.

Server security support includes standard Tomcat server functionality, the Spring Security Framework, plus OpenEdge security features and customizations.

For details, see the following topics:

- About customizing security for a production instance
- Enable ABL application authentication
- Disable HTTP in production instances
- Use TLS in PAS for OpenEdge
- About client authentication
- Configure Spring Security
- About managing users and remote access
- Authenticate using OpenEdge Authentication Gateway as a security token service
- Authenticate using OAuth2 and JWT
- Authenticate using SAML
- Extend OpenEdge SSO to web applications
- About support for server name indication

About customizing security for a production instance

You can customize the security settings on a production instance of PAS for OpenEdge. Some of the settings that you can customize are:

- **`manager` and `host-manager` web applications are undeployed by default**—The default Tomcat applications that enable remote online administration. In the PAS for OpenEdge Production product, when you create an instance using the `-f` option, these administration applications are not deployed. These applications are archived in the `$CATALINA_HOME/extras` directory and can be redeployed to enable their functionality.
- **Replacement of the default Tomcat `webapps/ROOT` application**—The replacement ROOT application specifically supports Progress applications including application security. The Tomcat ROOT application is archived in the `$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 extracts WAR files when they are legitimately deployed using PASMAN or standard Tomcat utilities.

- **Shutdown port disabled for UNIX and Windows**—The settings that 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**—The default access setting. PAS for OpenEdge includes JMX and JConsole support for instance management, but the default setting is for local access only.
- **Web crawler filtering enabled**—The setting that prevents instances from being overloaded by sessions initiated by web crawlers.
- **UNIX file permissions**—The default file permissions are initially only accessible and executable by ROOT users and groups. This is described in About user and file permissions on page 138.

Enable ABL application authentication

PAS for OpenEdge resolves authentication property settings by loading multiple `oeablSecurity.properties` files and using the first declared value it finds. The layered Spring Security property files are loaded in the following order:

1. *instance-name*`/webapps/`*webapp-name*`/WEB-INF/oeablSecurity.properties`
   - Properties and values applied to the web application
   - Matches the deployment configuration and supersedes the property values defined in *instance-name*`/conf/openedge.properties`
   - Can contain all or a subset of Spring Security properties
2. *instance-name*`/ablapps/`*abl-app-name*`/conf/oeablSecurity.properties`
   - Defaults applied to all web applications within a single ABL application
   - Matches the deployment configuration and supersedes the property values defined in *instance-name*`/conf/openedge.properties`
   - Can contain all or a subset of Spring Security properties
3. *instance-name*`/conf/oeablSecurity.properties`
   - Superset of Spring Security properties
   - Defaults applied to all web applications across all deployed ABL applications

Out of the box, the authentication mode of the production instance is set to anonymous, which is the lowest security level. The security level is set minimally because it is impossible to foresee which authentication implementation you require. Before running and deploying business applications to a production instance, increase the security level by enabling authentication in the staging environment.

The Spring Security framework, which is an integral part of PAS for OpenEdge, is adaptable to a wide variety of authentication models and architecture.

To enable authentication on a production instance:

1. Open the `oeablSecurity.properties` file at the security layer that you want to work on:
   - *instance-name*`/webapps/`*webapp-name*`/WEB-INF/oeablSecurity.properties`
   - *instance-name*`/ablapps/`*abl-app-name*`/conf/oeablSecurity.properties`
   - *instance-name*`/conf/oeablSecurity.properties`
2. Set the login model for the `client.login.model` property. The following HTTP client authentication login models are supported:

Login model	Description
`anonymous`	The default. All clients have public access. No direct logins or single sign-on (SSO) are allowed.
`basic`	Users authenticate using the HTTP BASIC header direct logins and SSO headers.
`form`	Users authenticate using a HTTP POST message and form data for direct login and SSO headers.
`container`	Users authenticate through the Tomcat realm services and authorize URL access using Spring Security.
`sso`	Users authenticate using CLIENT-PRINCIPAL access token through the OpenEdge SSO.
`oeauth2`	URLs are authorized through OpenEdge by validating OAuth2 JSON Web Tokens. For more information, see Authenticate using OAuth2 and JWT on page 140.
`saml`	Users authenticate and authorize using SAML tokens. For more information, see Authenticate using SAML on page 158.

3. Select the user account source for the `http.all.authmanager` property. The following user account sources are supported:

User account source	Description
`local`	The source is web application `WEB-INF/users.properties` with clear-text passwords. For more information, see About local and extended local user accounts on page 96.
`extlocal`	The source is the application `WEB-INF/users.properties` with encrypted passwords. For more information, see About local and extended local user accounts on page 96.

User account source	Description
`ldap`	The source is a simple LDAP (or Active Directory) server configuration. For more information, see About LDAP user accounts on page 98.
`oerealm`	The source is a bridge to the ABL application-maintained user accounts. For more information, see About OERealm user account configurations on page 99.

User account source	Description
`ad`	The source is a simple (constrained) Active Directory configuration. For more information, see About Active Directory user accounts on page 98.
`sts`	The source is the OpenEdge Authentication Gateway client. For more information, see *Learn about OpenEdge Authentication Gateway*

4. Update the default `ABLDomainRegistry.keystore` to include your domain name and domain access code:
   a. Design the OpenEdge domain name space to be used by the ABL application or instance, and assign a domain access code (DAC) to each domain name.
   b. Create an `ABLDomainRegistry.csv` 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
      ```

   c. Use the GENDOMREG utility to create an encrypted Java keystore file from the CSV file, for example:

      ```
      $DLC/bin/gendomreg ABLDomainRegistry.csv ABLDomainRegistry.keystore
      ```

   d. Copy the encrypted Java keystore file to the `.../conf` directory of each PAS for OpenEdge instance. Delete or override the default keystore.

      Note: If you use a non-default name for the keystore file, then change the `.registryFile` property in the `oeablSecurity.properties` file to the non-default keystore name.

   e. Design the role name pattern to designate an OpenEdge domain. For more information, see Obtain a domain name from a role name on page 121.
   f. After you establish a role name pattern, construct the Java regular expression (regex) pattern that distinguishes the domain name from any other normal user account granted roles.
   g. Set the `OEClientPrincipalFilter` bean's `domainRoleFilter` property to the value of the Java regex match pattern.
   h. If it is required by your security practice, set the `OEClientPrincipal.validateClientDomain` property in the `oeablSecurity.properties` to `true`. The `OEClientPrincipal.validateClientDomain` property is a Boolean value. When the value is set to true and when there is a `domainRoleFilter` defined, the property compares the domain specified by the client to the domain produced by the domain role filter. If the domains do not match, then the client request is rejected.

Note: Changing the `oeablSecurity-*.xml` file from local to another user account source, or adding and changing the users and passwords is not handled by the “local” `~/conf/tomcatusers.xml` file. You must modify the `oeablSecurity*.xml` file to access your authentication service.

About the 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 an ordered set of three values.

Note: Because an access control list (ACL) is an ordered set of three values, the format of an ACL does not fit well with a property file's name-value pairs. Therefore, URL access controls are not included in the oeablSecurity.properties file. Comma-separated value (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 that can include wildcards and regular expressions
- *method*—The HTTP access method
- *access*—The roles 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 code grants access to any user who has either ROLE_PSCAdmin or ROLE_PSCUser privileges to the data from a resource whose URL begins with /web/sales/.

```
"/web/sales/**", "GET", "hasAnyRole('ROLE_PSCAdmin','ROLE_PSCUser')"
```

About 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, to have a quick way of testing a web application's security structure. The local model allows you to 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
```

For more information, see the oeablSecurity.properties.README file.

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 (hashed/salted) passwords are generated with the GENSPRINGPWD utility.

For more information about the GENSPRINGPWD utility, see Generate encrypted passwords with GENSPRINGPWD on page 97.

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
```

For more information, see the `oeablSecurity.properties.README` file.

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 generate encrypted passwords, and add the secure passwords to the `users.properties` file. For more information about generating encrypted passwords, see Generate encrypted passwords with GENSPRINGPWD on page 97.

Generate encrypted passwords with GENSPRINGPWD

To provide greater security for user accounts stored in the `users.properties` file, 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 the only supported hashing algorithm.

GENSPRINGPWD syntax

```
genspringpwd password
```

password

A character string representing the password to be encrypted.

Store encrypted passwords

After you create 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 }
```

About 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, see the online LDAP administration documentation.

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 the `oeablSecurity.properties.README` file 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 uppercase because searches are case sensitive.

After configuring the LDAP model for your web application, you can implement CORS support.

About Active Directory user accounts

Microsoft Active Directory (AD) service provides a namespace for resolving a 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 the `oeablSecurity.properties.README` file 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.

About OERealm user account configurations

The OERealm authentication manager allows you to use an OpenEdge application server as a source of user account information for an 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 the OERealm model in the *OERealm user account server configuration* section of the `oeablSecurity.properties` file. See the `oeablSecurity.properties.README` file for more information.

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's 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, then no additional application server requests are performed.

- **OERealm AppServer ABL interface:** An ABL class that runs on the PAS for OpenEdge instance and responds to requests from the OERealm client and implements the IHybridRealm interface.

  The actions of the class are performed in the following order:

  1. Search for the user account using the name (*userid*[@*domain*]).
  2. Return the user account attributes for the user account.
  3. Validate the user account's password.

     Optionally, authorize the OERealm client and enable it to receive user account information.

Note: For more information about OERealm, see About OERealm security considerations on page 117.

Example configuration of OERealm authentication

The following Knowledge Base articule describes an end-to-end example of configuring OERealm authentication, and includes sample files that you can download and customize to run the example locally:

https://knowledgebase.progress.com/articles/Article/How-to-configure-OERealm-authentication-with-PASOE-on-OE-12-2

Create an OERealm service interface

To implement OpenEdge Single Point of Authentication (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 the PAS for OpenEdge instance to host the OERealm service interface class.
2. Create a singleton OOABL class that implements the OERealm HybridRealm interface.

   To obtain a sample class implementation of the HybridRealm, see the Sample HybridRealm.cls on page 103 topic.

   In the sample `HybridRealm.cls` file, the following methods are declared and defined:

   Table 2: 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 3: 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.

Sample HybridRealm.cls

This topic provides a sample `HybridRealm.cls` implementation that can be tailored to suit your needs.

```
 /*------------------------------------------------------------------------
    File        : HybridRealm
    Purpose     : Sample OOABL Server class of OERealm
    Syntax      :
    Description : Singleton class that implements IHybridRealm for use as
                  a single point of authentication for OE Realm Clients
                  viz. OEBPM, REST and Rollbase
    Author(s)   : Progress Software Corporation
    Created     : Thursday July 23 10:41:36 EST 2015
    Notes       : This sample uses OpenEdge database's _User table for illustration.
                  Users can use any other user account source,
  ----------------------------------------------------------------------*/

USING Progress.Lang.*.
USING Progress.Security.Realm.IHybridRealm.
USING OpenEdge.Security.Util.*.
USING OpenEdge.Security.Realm.OERealmDefs.

ROUTINE-LEVEL ON ERROR UNDO, THROW.

CLASS OpenEdge.Security.Realm.HybridRealm IMPLEMENTS IHybridRealm:

    DEFINE PRIVATE VARIABLE m_validateCP AS LOGICAL NO-UNDO INITIAL FALSE.
    DEFINE PRIVATE VARIABLE m_debugMsg AS LOGICAL NO-UNDO INITIAL FALSE.
    DEFINE PRIVATE VARIABLE m_passwd AS CHARACTER NO-UNDO.
    DEFINE PRIVATE VARIABLE m_role AS CHARACTER NO-UNDO.
    DEFINE PRIVATE VARIABLE m_autoUserNum  AS LOGICAL INITIAL YES NO-UNDO.
    DEFINE TEMP-TABLE ttUser LIKE _User.

    DEFINE PRIVATE VARIABLE m_spaProps AS CLASS OpenEdge.Security.Util.Properties.

    /*------------------------------------------------------------------------------
     Purpose: Default constructor
     Notes: Reads a property file names "spaservice.properties" to intialize:
         1. ValidateCP - If true, the OERealm methods will validate if the request came

                         from a trusted client by checking for existence of a
                         sealed client-Principal(C-P)
                       - If false, no such C-P validation will not be performed.
                       - This serves as an additional layer of security.It is highly
                         recommended that this validation be turned on
                       - In order to turn on this additional security validation:
                         1. The Implementor of OE Realm will create a .cp file using
                         genspacp utility and distribute it to the trusted OERealm
Clients.
-----------------------------------------------------------------------
The syntax of genspacp is as below:
genspacp -password <text> [-user <text> -role <text> -domain <text> -file <text>]
-----------------------------------------------------------------------
                       2. The .cp file will be sealed by the "password" field provided

                          to the utility
         2. Password  -  The value will be used to validate the incoming C-P
                      -  It must match the value provided while generating .cp using
                         genspacp utility
                      -  It is applicable only when validateCP is true
         3. Role      -  The value must match the role that the incoming C-P contains
                      -  It must match the value provided while generating .cp using
                         genspacp utility
                      -  It is applicable only when validateCP is true
         4. DebugMsg  - If true, debug messages will be logged in server log file
    ------------------------------------------------------------------------------*/

    CONSTRUCTOR PUBLIC HybridRealm (  ):
        SUPER ().
        m_spaProps = NEW Properties("spaservice.properties").
        m_validateCP = m_spaProps:GetLogicalProperty("validateCP", ?).
        m_debugMsg = m_spaProps:GetLogicalProperty("DebugMsg", ?).
```

```
        IF m_validateCP THEN DO:
            m_passwd = m_spaProps:GetCharacterProperty("Password", "").
            m_role = m_spaProps:GetCharacterProperty("role", "SpaClient").
        END.
        IF m_debugMsg THEN DO:
            MESSAGE "Loaded property file spaservice.properties".
            MESSAGE "   DebugMsg: " m_passwd.
            IF m_validateCP THEN DO:
                /* Needed only when validateCP is true */
                MESSAGE "   Password (to be used to validate the incoming C-P): "
m_passwd.
                MESSAGE "   Role: " m_role.
            END.
        END.

    END CONSTRUCTOR.


    /*------------------------------------------------------------------------------
     Purpose: Returns true if the C-P received in the request is valid, else returns
false
     Notes: Uses "password" value configured in spaservice.properties to validate the
C-P
    ------------------------------------------------------------------------------
*/
    METHOD PROTECTED LOGICAL ValidateClient(  ):

        DEFINE VARIABLE result AS LOGICAL NO-UNDO INITIAL FALSE.
        DEFINE VARIABLE hCP AS HANDLE NO-UNDO.
        DEFINE VARIABLE i AS INTEGER NO-UNDO.

        hCP = SESSION:CURRENT-REQUEST-INFO:GetClientPrincipal().
        IF (? <> hCP) THEN
        DO:
            MESSAGE "Request contained a C-P".
            result = hCP:VALIDATE-SEAL (m_passwd).
            MESSAGE "Validation with seal: " m_passwd "succeeded".
        END.
        ELSE
            MESSAGE "Request did not contain a C-P".

        IF result = TRUE THEN
        DO:
            IF hCP:ROLES <> m_role THEN
                result = FALSE.
        END.

        RETURN result.

    END METHOD.

    /*------------------------------------------------------------------------------
     Purpose: Returns the attribute value of a user
     Notes:
     INPUT : theUserId - positive user number
                       - In this example it corresponds to _User._User_number.
             attrName  - String name for an attribute to be fetched
                       - Refer to OERealmDefs for the attribute name that this
                         sample supports
     Throws:
     Progress.Lang.AppError("Unauthorized client", 1).
     Progress.Lang.AppError("Invalid User Id")
    ------------------------------------------------------------------------------*/

    METHOD PUBLIC CHARACTER GetAttribute( INPUT theUserId AS INTEGER, INPUT attrName
AS CHARACTER ):

        DEFINE VARIABLE retVal AS CHARACTER NO-UNDO.
        DEFINE VARIABLE fqUserid    AS CHARACTER NO-UNDO.
```

```
        IF m_validateCP THEN DO:
            IF THIS-OBJECT:ValidateClient() = FALSE THEN
                UNDO, THROW NEW Progress.Lang.AppError("Unauthorized client", 1).
        END.

        IF m_debugMsg THEN
            MESSAGE "Attempting to get Attribute " attrName " for user number "
theUserId.
        FIND FIRST _User WHERE _User._User_number = theUserId NO-ERROR.
        IF AVAILABLE _User THEN DO:
            IF ( _User._Domain-Name = "" ) THEN
                fqUserid = _User._Userid.
            ELSE
                fqUserid = _User._Userid + "@" + _User._Domain-Name.

            CASE attrName:
                WHEN OERealmDefs:REALM_ATTR_SURNAME THEN
                    retVal = _User._Surname.
                WHEN OERealmDefs:REALM_ATTR_GIVENNAME THEN
                    retVal = _User._Given_name.
                WHEN OERealmDefs:REALM_ATTR_TELEPHONE THEN
                    retVal = _User._Telephone.
                WHEN OERealmDefs:REALM_ATTR_EMAIL THEN
                    retVal = _User._Email.
                WHEN OERealmDefs:REALM_ATTR_DESCRIPTION THEN
                    retVal = _User._Description.
                WHEN OERealmDefs:REALM_ATTR_USERID THEN
                    retVal = fqUserid.
                WHEN OERealmDefs:REALM_ATTR_DOMAIN THEN
                    retVal = _User._Domain-Name.
                WHEN OERealmDefs:REALM_ATTR_ENABLED THEN DO:
                    IF ( ? = _User._Disabled OR _User._Disabled = NO ) THEN
                       retVal = "1".      /* Enabled */
                    ELSE
                       retVal = "0".     /* !Enabled */
                END.
                WHEN OERealmDefs:REALM_ATTR_ROLES THEN DO:
                    DEFINE VARIABLE roles   AS CHARACTER NO-UNDO.
                    DEFINE VARIABLE count   AS  INTEGER INITIAL 0 NO-UNDO.

                    IF m_debugMsg THEN
                      MESSAGE "Get user roles for user number " theUserId "(" fqUserid
 ")".
                    roles = ?.
                    FOR EACH _sec-granted-role WHERE _sec-granted-role._grantee =
fqUserid :
                        IF ( count = 0 ) THEN
                            roles = "".
                        ELSE
                            roles = roles + ",".
                        roles = roles + _sec-granted-role._Role-name.
                        count = count + 1.
                    END.
                    IF m_debugMsg THEN
                        MESSAGE "Found '" roles "' roles for user-id" theUserId "("
fqUserid ")".
                    retVal = roles.
                END.
                OTHERWISE DO:
                    IF m_debugMsg THEN
                        MESSAGE "Unknown attribute name in GetAttribute:" attrName.
                    retVal = ?.
                END.
            END.
        END.
        ELSE DO:
            IF m_debugMsg THEN
                MESSAGE "Unknown user-id found in GetAttribute:" theUserId.
```

```
            UNDO, THROW NEW Progress.Lang.AppError("Invalid User Id").
        END.

        RELEASE _User.
        IF m_debugMsg THEN
            MESSAGE "Returning " retVal " for attribute value".
        RETURN retVal.

    END METHOD.

    /*------------------------------------------------------------------------------
     Purpose:Returns all the attributes defined for a user as a comma separeted list
     Notes:
     INPUT : theUserId - positive user number
                       - In this example it corresponds to _User._User_number.
     Throws:
     Progress.Lang.AppError("Unauthorized client", 1).
     Progress.Lang.AppError("User not found").
    ------------------------------------------------------------------------------*/
    METHOD PUBLIC CHARACTER EXTENT GetAttributeNames( INPUT theUserId AS INTEGER ):

        DEFINE VARIABLE results AS CHARACTER EXTENT NO-UNDO.
        DEFINE VARIABLE attrNames AS CHARACTER EXTENT 10 NO-UNDO.
        DEFINE VARIABLE attrVal AS CHARACTER NO-UNDO.
        DEFINE VARIABLE numAttrs AS INTEGER NO-UNDO INITIAL 0.
        DEFINE VARIABLE i AS INTEGER NO-UNDO.
        DEFINE VARIABLE fqUserid    AS CHARACTER NO-UNDO.

        IF m_validateCP THEN DO:
            IF THIS-OBJECT:ValidateClient() = FALSE THEN
                UNDO, THROW NEW Progress.Lang.AppError("Unauthorized client", 1).
        END.
        IF m_debugMsg THEN
            MESSAGE "Getting attributes for user number " theUserId.
        FIND FIRST _User WHERE _User._User_number = theUserId NO-ERROR.
        IF AVAILABLE _User THEN DO:
            IF ( _User._Domain-Name = "" ) THEN
                fqUserid = _User._Userid.
            ELSE
                fqUserid = _User._Userid + "@" + _User._Domain-Name.

            /* Check to see what attributes have values */
            attrVal = _User._Surname.
            IF attrVal <> ? THEN DO:
                 numAttrs = numAttrs + 1.
                 attrNames[numAttrs] = OERealmDefs:REALM_ATTR_SURNAME.
            END.
            attrVal = _User._Given_name.
            IF attrVal <> ? THEN DO:
                numAttrs = numAttrs + 1.
                 attrNames[numAttrs] = OERealmDefs:REALM_ATTR_GIVENNAME.
            END.
            attrVal = _User._Telephone.
            IF attrVal <> ? THEN DO:
                numAttrs = numAttrs + 1.
                 attrNames[numAttrs] = OERealmDefs:REALM_ATTR_TELEPHONE.
            END.
            attrVal = _User._Email.
            IF attrVal <> ? THEN DO:
                numAttrs = numAttrs + 1.
                 attrNames[numAttrs] = OERealmDefs:REALM_ATTR_EMAIL.
            END.
            attrVal = _User._Description.
            IF attrVal <> ? THEN DO:
                numAttrs = numAttrs + 1.
                 attrNames[numAttrs] = OERealmDefs:REALM_ATTR_DESCRIPTION.
            END.
            attrVal = _User._Userid.
            IF attrVal <> ? THEN DO:
```

```
               numAttrs = numAttrs + 1.
                 attrNames[numAttrs] = OERealmDefs:REALM_ATTR_USERID.
            END.
            attrVal = _User._Domain-Name.
            IF attrVal <> ? THEN DO:
               numAttrs = numAttrs + 1.
                 attrNames[numAttrs] = OERealmDefs:REALM_ATTR_DOMAIN.
            END.
            DO:
              FIND FIRST _sec-granted-role WHERE _sec-granted-role._grantee = fqUserid
NO-ERROR.
                IF AVAILABLE _sec-granted-role THEN DO:
                    numAttrs = numAttrs + 1.
                    attrNames[numAttrs] = OERealmDefs:REALM_ATTR_ROLES.
                END.
            END.

            IF numAttrs > 0 THEN DO:
                IF m_debugMsg THEN
                    MESSAGE "Number of attribute names: " numAttrs.
                EXTENT(results) = numAttrs.
                DO i = 1 TO numAttrs:
                    results[i] = attrNames[i].
                    IF m_debugMsg THEN
                        MESSAGE results[i].
                END.
            END.
        END.
        ELSE
            UNDO, THROW NEW Progress.Lang.AppError("User not found").
        RELEASE _User.
        RETURN results.

    END METHOD.
    /*------------------------------------------------------------------------------
     Purpose:Returns all the user names as character extent
     Notes:
    ------------------------------------------------------------------------------*/
    METHOD PUBLIC CHARACTER EXTENT GetUserNames(  ):

        DEFINE VARIABLE results AS CHARACTER EXTENT NO-UNDO.
        DEFINE VARIABLE num AS INTEGER NO-UNDO INITIAL 0.

        IF m_validateCP THEN DO:
            IF THIS-OBJECT:ValidateClient() = FALSE THEN
                UNDO, THROW NEW Progress.Lang.AppError("Unauthorized client", 1).
        END.

        /* First count the names */
        FOR EACH _User:
            num = num + 1.
        END.

        EXTENT(results) = num.
        num = 1.
        FOR EACH _User:
            IF ( _User._Domain-Name = "" ) THEN
                results[num] = _User._Userid.
            ELSE
                results[num] = _User._Userid + "@" + _User._Domain-Name.
            num = num + 1.
        END.

        RELEASE _User.
        RETURN results.

    END METHOD.

    /*------------------------------------------------------------------------------
```

```
     Purpose: Returns the fully qualified usernames a matching a query pattern
     Notes:
     INPUT : queryString - character value which against which the userid should be
matched
    ------------------------------------------------------------------------------*/
    METHOD PUBLIC CHARACTER EXTENT GetUserNamesByQuery( INPUT queryString AS CHARACTER
 ):

        DEFINE VARIABLE results AS CHARACTER EXTENT NO-UNDO.
        DEFINE VARIABLE num AS INTEGER NO-UNDO INITIAL 0.

        IF m_validateCP THEN DO:
            IF THIS-OBJECT:ValidateClient() = FALSE THEN
                UNDO, THROW NEW Progress.Lang.AppError("Unauthorized client", 1).
        END.

        IF m_debugMsg THEN
            MESSAGE "QueryString = " queryString.

        /* First count the names */
        FOR EACH _User WHERE _User._Userid MATCHES queryString:
            IF m_debugMsg THEN
                MESSAGE "Match: " _User._Userid + "@" + _User._Domain-Name.
            num = num + 1.
        END.

        IF num = 0 THEN DO:
            IF m_debugMsg THEN
                MESSAGE "No users found".
            RETURN results.
        END.

        EXTENT(results) = num.
        num = 0.
        FOR EACH _User WHERE _User._Userid MATCHES queryString:
            num = num + 1.
            IF ( _User._Domain-Name = "" ) THEN
                results[num] = _User._Userid.
            ELSE
                results[num] = _User._Userid + "@" + _User._Domain-Name.
        END.
        IF m_debugMsg THEN
            MESSAGE "Found " num " users".

        RELEASE _User.
        RETURN results.


    END METHOD.

    /*------------------------------------------------------------------------------
     Purpose: Returns the fully qualified usernames whose attributes match
     Notes:
     INPUT : attrName  - character value representing the attribute name
             attrValue - character value representing the attribute value
    ------------------------------------------------------------------------------*/
    METHOD PUBLIC CHARACTER EXTENT GetUserNamesByQuery( INPUT attrName AS CHARACTER,
INPUT attrValue AS CHARACTER ):

        DEFINE VARIABLE results AS CHARACTER EXTENT NO-UNDO.
        DEFINE VARIABLE temp AS CHARACTER EXTENT 100 NO-UNDO.
        DEFINE VARIABLE num AS INTEGER NO-UNDO INITIAL 0.
        DEFINE VARIABLE whichAttr AS CHARACTER NO-UNDO.
        DEFINE VARIABLE attrCounter AS INTEGER NO-UNDO.

        IF m_validateCP THEN DO:
            IF THIS-OBJECT:ValidateClient() = FALSE THEN
                UNDO, THROW NEW Progress.Lang.AppError("Unauthorized client", 1).
        END.
```

```
        IF m_debugMsg THEN
            MESSAGE "Attribute name: " attrName " Attribute value: " attrValue.

        /* First count the names */
        FOR EACH _User:
            CASE whichAttr:
                WHEN OERealmDefs:REALM_ATTR_SURNAME THEN
                    IF _User._Surname MATCHES attrValue THEN DO:
                        IF m_debugMsg THEN
                           MESSAGE "Match: " _User._Userid + "@" + _User._Domain-Name.

                        num = num + 1.
                        IF ( _User._Domain-Name = "" ) THEN
                            results[num] = _User._Userid.
                        ELSE
                            temp[num] = _User._Userid + "@" + _User._Domain-Name.
                    END.
                WHEN OERealmDefs:REALM_ATTR_GIVENNAME THEN
                    IF _User._Given_name MATCHES attrValue THEN DO:
                        IF m_debugMsg THEN
                           MESSAGE "Match: " _User._Userid + "@" + _User._Domain-Name.

                        num = num + 1.
                        IF ( _User._Domain-Name = "" ) THEN
                            results[num] = _User._Userid.
                        ELSE
                            temp[num] = _User._Userid + "@" + _User._Domain-Name.
                    END.
                WHEN OERealmDefs:REALM_ATTR_TELEPHONE THEN
                    IF _User._Telephone MATCHES attrValue THEN DO:
                        IF m_debugMsg THEN
                           MESSAGE "Match: " _User._Userid + "@" + _User._Domain-Name.

                        num = num + 1.
                        IF ( _User._Domain-Name = "" ) THEN
                            results[num] = _User._Userid.
                        ELSE
                            temp[num] = _User._Userid + "@" + _User._Domain-Name.
                    END.
                WHEN OERealmDefs:REALM_ATTR_EMAIL THEN
                    IF _User._Email MATCHES attrValue THEN DO:
                        IF m_debugMsg THEN
                            MESSAGE "Match: " _User._Userid.
                        num = num + 1.
                        IF ( _User._Domain-Name = "" ) THEN
                            results[num] = _User._Userid.
                        ELSE
                            temp[num] = _User._Userid + "@" + _User._Domain-Name.
                    END.
                WHEN OERealmDefs:REALM_ATTR_DESCRIPTION THEN
                    IF _User._Description MATCHES attrValue THEN DO:
                        IF m_debugMsg THEN
                           MESSAGE "Match: " _User._Userid + "@" + _User._Domain-Name.

                        num = num + 1.
                        IF ( _User._Domain-Name = "" ) THEN
                            results[num] = _User._Userid.
                        ELSE
                            temp[num] = _User._Userid + "@" + _User._Domain-Name.
                    END.
            END.
        END.

        IF num = 0 THEN DO:
            IF m_debugMsg THEN
                MESSAGE "No users found".
            RETURN results.
        END.
```

```
        EXTENT(results) = num.
        DO attrCounter = 1 TO num:
            results[attrCounter] = temp[attrCounter].
        END.
        IF m_debugMsg THEN
            MESSAGE "Found " num " users".

        RELEASE _User.
        RETURN results.


    END METHOD.

    /*------------------------------------------------------------------------------
     Purpose: Removes an attribute from User account based on unique user number
     Notes:
     INPUT : theUserId - positive user number
                       - In this example it corresponds to _User._User_number.
              attrName  - String representing the attribute name to be removed
                       - Refer to OERealmDefs for the attribute name that this
                         sample supports
     Throws:
     Progress.Lang.AppError("Unauthorized client", 1).
     Progress.Lang.AppError("Invalid User Id").
    ------------------------------------------------------------------------------*/
    METHOD PUBLIC LOGICAL RemoveAttribute( INPUT theUserId AS INTEGER, INPUT attrName
AS CHARACTER ):

        DEFINE VARIABLE retVal AS LOGICAL NO-UNDO INITIAL TRUE.

        IF m_validateCP THEN DO:
            IF THIS-OBJECT:ValidateClient() = FALSE THEN
                UNDO, THROW NEW Progress.Lang.AppError("Unauthorized client", 1).
        END.

        IF m_debugMsg THEN
            MESSAGE "Attempting to remove Attribute " attrName " for user number "
theUserId.
        FIND FIRST _User WHERE _User._User_number = theUserId NO-ERROR.
        IF AVAILABLE _User THEN DO:
            CASE attrName:
                WHEN OERealmDefs:REALM_ATTR_SURNAME THEN
                    _User._Surname = ?.
                WHEN OERealmDefs:REALM_ATTR_GIVENNAME THEN
                    _User._Given_name = ?.
                WHEN OERealmDefs:REALM_ATTR_TELEPHONE THEN
                    _User._Telephone = ?.
                WHEN OERealmDefs:REALM_ATTR_EMAIL THEN
                    _User._Email = ?.
                WHEN OERealmDefs:REALM_ATTR_DESCRIPTION THEN
                    _User._Description = ?.
                OTHERWISE DO:
                    retVal = FALSE.
                    RELEASE _User.
                END.
            END.
        END.
        ELSE
            UNDO, THROW NEW Progress.Lang.AppError("Invalid User Id").

        RELEASE _User.
        RETURN retVal.

    END METHOD.

    /*------------------------------------------------------------------------------
     Purpose: Sets an attribute value for a user
     value
```

```
     Notes:
     INPUT : theUserId - positive user number
                       - In this example it corresponds to _User._User_number.
             attrName  - String representing the attribute name to be removed
                       - Refer to OERealmDefs for the attribute name that this
                         sample supports
             attrValue - value of an attribute
     Throws:
     Progress.Lang.AppError("Unauthorized client", 1).
     Progress.Lang.AppError("Invalid User Id").
    ------------------------------------------------------------------------------*/
    METHOD PUBLIC LOGICAL SetAttribute( INPUT theUserId AS INTEGER, INPUT attrName AS
CHARACTER, INPUT attrValue AS CHARACTER ):

        DEFINE VARIABLE retVal AS LOGICAL NO-UNDO INITIAL TRUE.

        IF m_validateCP THEN DO:
            IF THIS-OBJECT:ValidateClient() = FALSE THEN
                UNDO, THROW NEW Progress.Lang.AppError("Unauthorized client", 1).
        END.

        IF m_debugMsg THEN
           MESSAGE "Attempting to set Attribute " attrName " for user number " theUserId
 " with a value of " attrValue.
        FIND FIRST _User WHERE _User._User_number = theUserId NO-ERROR.
        IF AVAILABLE _User THEN DO:
            CASE attrName:
                WHEN OERealmDefs:REALM_ATTR_SURNAME THEN
                    _User._Surname = attrValue.
                WHEN OERealmDefs:REALM_ATTR_GIVENNAME THEN
                    _User._Given_name = attrValue.
                WHEN OERealmDefs:REALM_ATTR_TELEPHONE THEN
                    _User._Telephone = attrValue.
                WHEN OERealmDefs:REALM_ATTR_EMAIL THEN
                    _User._Email = attrValue.
                WHEN OERealmDefs:REALM_ATTR_DESCRIPTION THEN
                    _User._Description = attrValue.
                WHEN OERealmDefs:REALM_ATTR_PASSWD THEN DO:
                    MESSAGE "Attempting to set Attribute " attrName " for user number
" theUserId " with a value of " attrValue.
                    FIND FIRST _User WHERE _User._User_number = theUserId NO-ERROR.
                    IF AVAILABLE _user THEN DO:
                       BUFFER-COPY _user to ttUser.
                       ttUser._password = ENCODE(attrValue).
                       DELETE _User.
                      BUFFER-COPY ttUser EXCEPT _TenantId TO _User.
                       RELEASE _User.
                       MESSAGE "Successfully  changed password for user with id "
theUserId.
                       RETURN true.
                    END.
                    ELSE DO:
                        MESSAGE "User " theUserId " not found".
                        RETURN false.
                    END.
                END.
                OTHERWISE DO:
                    retVal = FALSE.
                    RELEASE _User.
                END.
            END.
        END.
        ELSE
            UNDO, THROW NEW Progress.Lang.AppError("Invalid User Id").

        RELEASE _User.
        RETURN retVal.

    END METHOD.
```

```
    /*------------------------------------------------------------------------------
     Purpose: Sets an password for a given user
     Notes:
     INPUT : theUserId - positive user number
                       - In this example it corresponds to _User._User_number.
             password  - String value to be set as the password
     The incoming request must contain a C-P to perform this operation
     Throws:
     Progress.Lang.AppError("Unauthorized client", 1).
     Progress.Lang.AppError("Invalid User Id").
    ------------------------------------------------------------------------------*/
   METHOD PUBLIC LOGICAL SetAttribute( INPUT theUserId AS INTEGER,  INPUT password AS
 CHARACTER ):

        DEFINE VARIABLE retVal AS LOGICAL NO-UNDO INITIAL TRUE.

        IF THIS-OBJECT:ValidateClient() = FALSE THEN
            UNDO, THROW NEW Progress.Lang.AppError("Unauthorized client", 1).

        IF m_debugMsg THEN
            MESSAGE "Attempting to set password for user number " theUserId " with a
value of xxxx".
        FIND FIRST _User WHERE _User._User_number = theUserId NO-ERROR.
        IF AVAILABLE _User THEN
        DO:
            _User._Password = Encode(password).
        END.
        ELSE
            UNDO, THROW NEW Progress.Lang.AppError("Invalid User Id").

        RELEASE _User.
        RETURN retVal.

   END METHOD.



   /*------------------------------------------------------------------------------
     Purpose: validate a password for a given user
     Notes:
     INPUT : theUserId - positive user number
                       - In this example it corresponds to _User._User_number.
             password  - String value to be set as the password
     Throws:
     Progress.Lang.AppError("Unauthorized client", 1).
     Progress.Lang.AppError("Invalid User Id").
    ------------------------------------------------------------------------------*/
   METHOD PUBLIC LOGICAL ValidatePassword( INPUT theUserId AS INTEGER, INPUT password
 AS CHARACTER ):

        DEFINE VARIABLE retVal AS LOGICAL NO-UNDO INITIAL FALSE.

        IF m_validateCP THEN DO:
            IF THIS-OBJECT:ValidateClient() = FALSE THEN
                UNDO, THROW NEW Progress.Lang.AppError("Unauthorized client", 1).
        END.

        IF m_debugMsg THEN DO:
            MESSAGE "Validate password using basic".
            MESSAGE "Password: " password.
        END.

        FIND FIRST _User WHERE _User._User_number = theUserId NO-ERROR.
        IF AVAILABLE _User THEN DO:
            IF _User._Password = ENCODE(password) THEN
                retVal = TRUE.
            ELSE
                RETURN FALSE.
```

```
        END.
        ELSE
            UNDO, THROW NEW Progress.Lang.AppError("Invalid User Id").

        RELEASE _User.
        RETURN retVal.

    END METHOD.

    /*------------------------------------------------------------------------------
     Purpose: validate a password for a given user using digest
     Notes:
     INPUT : theUserId - positive user number
                       - In this example it corresponds to _User._User_number.
             digest    - String representing the digest value of the password
             nounce    - String representing the nounce
             timestamp - String representing the timestamp.

     NOTE: Digest version of ValidatePassword requires that the passwords be
     stored in such a way that they can be retrieved as clear text.
     Since, the _password field of _User table stores password in one way
     encoded format from which the clear-text password cannot be retrieved,
     therefore, in this sample, we have encrypted the passwords using
     db/encPwd.p and stored  in _User._U-misc2[1].
     The  encrypted passwords stored thus in the _User._U-misc2[1] can be
     retrieved as clear-text using db/decPwd.p.
     Please note that this is just a sample implementation, you can store the
     passwords in any other way of your choice such that it can be recovered
     as a clear-text.

     Throws:
     Progress.Lang.AppError("Unauthorized client", 1).
     Progress.Lang.AppError("User not found").
     Progress.Lang.AppError("Old timestamp sent").
    ------------------------------------------------------------------------------*/
    METHOD PUBLIC LOGICAL ValidatePassword( INPUT theUserId AS INTEGER, INPUT digest
AS CHARACTER, INPUT nonce AS CHARACTER, INPUT timestamp AS CHARACTER ):

        DEFINE VARIABLE retVal AS LOGICAL NO-UNDO INITIAL FALSE.
        DEFINE VARIABLE myDigest AS CHARACTER NO-UNDO.
        DEFINE VARIABLE df AS CHARACTER NO-UNDO.
        DEFINE VARIABLE remoteTS AS DATETIME-TZ NO-UNDO.
        DEFINE VARIABLE dif AS INT64 NO-UNDO.
        DEFINE VARIABLE hash AS RAW NO-UNDO.

        DEFINE VARIABLE clrTxtPwd AS CHARACTER NO-UNDO.
        DEFINE VARIABLE userName AS CHARACTER NO-UNDO.
        DEFINE VARIABLE domain AS CHARACTER NO-UNDO.

        IF m_validateCP THEN DO:
            IF THIS-OBJECT:ValidateClient() = FALSE THEN
                UNDO, THROW NEW Progress.Lang.AppError("Unauthorized client", 1).
        END.

        FIND FIRST _User WHERE _User._User_number = theUserId NO-ERROR.
        IF AVAILABLE _User THEN DO:
           userName = _User._Userid.
           IF (_User._Domain-name <> '') THEN
              userName = userName + "@" + _User._Domain-name.
           IF m_debugMsg THEN
              MESSAGE "Decoding password for " userName.
           IF (_User._U-misc2[1] <> ? ) THEN
               run db/decPwd.p (userName, _User._U-misc2[1], output clrTxtPwd ).
           ELSE
           DO:
               MESSAGE "Encrypted password not stored in _U-misc2[1] " .
               /* Since encypted password is not stored in _U-misc2[1]
               thus assuming that _Password stores clear-text password
               which should ideally not be the case */
```

```
            clrTxtPwd = _User._Password.
         END.
      END.
      ELSE
        UNDO, THROW NEW Progress.Lang.AppError("User not found").

      RELEASE _User.

      IF m_debugMsg THEN DO:
          MESSAGE "Validate password using digest".
          MESSAGE "Digest: " digest.
          MESSAGE "Nonce: " nonce.
          MESSAGE "Timestamp: " timestamp.
      END.

      df = SESSION:DATE-FORMAT.
      SESSION:DATE-FORMAT = "ymd".
      remoteTS = DATETIME-TZ(timestamp).
      SESSION:DATE-FORMAT = df.

      dif = ABSOLUTE(INTERVAL(NOW, remoteTS, "seconds")).

      IF dif > 1 THEN
        UNDO, THROW NEW Progress.Lang.AppError("Old timestamp sent").

      myDigest = clrTxtPwd + nonce + timestamp.
      hash = SHA1-DIGEST(myDigest).
      myDigest = BASE64-ENCODE(hash).
      IF m_debugMsg THEN
          MESSAGE "Computed digest: " myDigest.
      IF myDigest = digest THEN
          retVal = TRUE.
      ELSE
          retVal = FALSE.
      RETURN retVal.

  END METHOD.

  /*------------------------------------------------------------------------------
   Purpose: validate if the given user exists and if it exists returns the user number

   Notes:
   INPUT : userName - Fully qualified user name
   Throws:
   Progress.Lang.AppError("Unauthorized client", 1).
   Progress.Lang.AppError("User not found", 8).
   Progress.Lang.AppError("User number unknown for the user " + userName).
  ------------------------------------------------------------------------------*/
  METHOD PUBLIC INTEGER ValidateUser( INPUT userName AS CHARACTER ):

      DEFINE VARIABLE userNum AS INTEGER NO-UNDO.
      DEFINE VARIABLE usernm  AS CHARACTER INITIAL "" NO-UNDO.
      DEFINE VARIABLE domain  AS CHARACTER INITIAL "" NO-UNDO.
      usernm = ENTRY(1,userName,"@").
      domain = ENTRY(2,userName,"@") NO-ERROR.

      IF m_debugMsg THEN
          MESSAGE "Validating user " + userName.

      IF m_validateCP THEN DO:
          IF THIS-OBJECT:ValidateClient() = FALSE THEN
              UNDO, THROW NEW Progress.Lang.AppError("Unauthorized client", 1).
      END.
      FIND FIRST _User WHERE _User._Userid = usernm AND
                             _User._Domain-Name = domain AND
                             _User._sql-only-user = NO NO-ERROR.
      IF AVAILABLE _User THEN DO:
          userNum = _User._User_number.
          IF ( ? = userNum AND m_autoUserNum ) THEN DO:
```

```
                IF m_debugMsg THEN
                    MESSAGE "User number unknown for the user " + userName.

                /* NOTE: In this sample we throw
               * Progress.Lang.AppError("User number unknown for the user " + userName)

                 * when the userNum is UNKNOWN. You may choose not to throw an error
and
                 * instead assign a unique numeric number while validating the user.
For
                 * this, uncomment the below code to assign a unique id by using a
                 * Sequence named Next-User-Num. */
                /*
                DEFINE VARIABLE newUserNum      AS INTEGER  NO-UNDO.
                newUserNum = NEXT-VALUE( Next-User-Num ).
                MESSAGE "Lookup user account issued user" userName "ID number"
newUserNum.
                assign _User._User_number = newUserNum.
                userNum = _User._User_number.
                */
                UNDO, THROW NEW Progress.Lang.AppError("User number unknown for the
user " + userName).
            END.
            IF m_debugMsg THEN
                MESSAGE "Returning user num " + STRING(userNum) + " for user " +
userName.
            RELEASE _User.
            RETURN userNum.
        END.
        ELSE
          UNDO, THROW NEW Progress.Lang.AppError("User not found", 8).
    END METHOD.

END CLASS.
```

About OERealm security considerations

The OERealm authentication process must be secure against unwarranted attacks.

The following considerations must be observed for security purposes:

- OERealm service interace security
- OERealmAuthProvider security
- Prevent authorization before authentication prevention

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 CLIENT-PRINCIPAL that the OERealm service is capable of validating.

Because the OERealm user details service can be configured to send a sealed CLIENT-PRINCIPAL to the PAS for OpenEdge instance OERealm service interface on each method it invokes, the OERealm service interface can incorrectly intercept the CLIENT-PRINCIPAL 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, Progress recommend that you use the `CLIENT-PRINCIPAL:VALIDATE-SEAL()` method along with encoded domain access codes (`"oech1::xxxx"`) to validate the CLIENT-PRINCIPAL. 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 CLIENT-PRINCIPAL-related activities, such as a session, database, server connection, or BPM identity.

OERealmAuthProvider security

You can configure the OpenEdge domain access code to seal the CLIENT-PRINCIPAL in the single sign-on (SSO) mode. By default, the Spring Security `OERealmAuthProvider` bean creates the CLIENT-PRINCIPAL as a single tenant. The problem is that every web application user of the CLIENT-PRINCIPAL 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 CLIENT-PRINCIPAL:

  - 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 4: 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 CLIENT-PRINCIPAL objects in multi-tenant applications. This file contains a list of domains that are allowed to have access to the CLIENT-PRINCIPAL.

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 OpenEdge domains that are allowed to have access to the CLIENT-PRINCIPAL 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 OpenEdge REST Web application's /WEB-INF/classes directory.
3. Configure the OERealmAuthProvider bean to load and sign a CLIENT-PRINCIPAL.

Prevent authorization before authentication

The CLIENT-PRINCIPAL 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 unnecessary client access by using the CLIENT-PRINCIPAL 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 CLIENT-PRINCIPAL just like it is used in other parts of the ABL application.

To develop the web application's OERealmUserDetails to act as an OpenEdge application server client:

1. Use the genspacp utility in $DLC/bin, to create a sealed CLIENT-PRINCIPAL that represents a OERealm service client. The utility requires the user ID, domain name, domain access code, and output filename to create a binary file that contains the sealed CLIENT-PRINCIPAL 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 $DLC/bin 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 CLIENT-PRINCIPAL validation code. The code can be in the PAS for OpenEdge instance activate procedure or as step in each of the OERealm class methods. Progress recommends 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 CLIENT-PRINCIPAL in the created binary output file, the following sequence occurs:

1. After the web application is loaded, if the OERealm.UserDetails.realmTokenFile property contains the output filename, the file is used to load the CLIENT-PRINCIPAL. 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 CLIENT-PRINCIPAL to the PAS for OpenEdge instance OERealm service interface methods.
3. The OERealm service interface validates the CLIENT-PRINCIPAL. The following validation outcomes are possible:
   - If the CLIENT-PRINCIPAL is not sent to the interface, then an error is returned.

- If the CLIENT-PRINCIPAL is sent to the interface, then the interface is used to validate the domain access code. If the validation fails, then an error is returned.
- If the CLIENT-PRINCIPAL validation is successful, then the execution of the ABL class method continues.

About multi-domain support for user accounts

PAS for OpenEdge can extend 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 provides 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 can 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 that holds 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 about multi-domain support for authentication systems like LDAP, see the following topics:

- Resolve user ID and domain names on page 120
- Obtain a domain name from a role name on page 121
- Configure multiple domains on page 122
- About OpenEdge domains in LDAP configurations on page 123

Resolve 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 and domain name), then go to Step 3.
2. Resolve the domain name by doing the following:

a. Look for a user role in the `OEClientPrincipalFilter.domainRoleFilter` property, which is recognized as holding a domain name. If the domain name is found, then go to Step 3.

Note: All `OEClientPrincipalFilter` properties are set in the `oeablSecurity.properties` file.

---

b. Use the `OEClientPrincipalFilter.domain` property value. If the domain name is found, then go to Step 3.

c. If the domain name remains unresolved after Steps 2a and 2b, then raise an error.

3. Create a CLIENT-PRINCIPAL object and populate its data fields.
4. If the `OEClientPrincipalFilter.key` property is configured, then 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, then raise an error.
7. Seal the CLIENT-PRINCIPAL with the DAC, and store it in the user's HTTP login session context to be used in later client requests.

Obtain 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 regular expression (regex) pattern that identifies role names produced by Spring Security and extracts 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 can be any valid pattern with a single extraction group (a Java regex pattern enclosed with parentheses). The full pattern is used to match a role name. Anything that matches the pattern enclosed with 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. Because the role names presented to the OEClientPrincipalFilter bean are in the form `ROLE_`*`user-role-name`*, this is what is 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.

Configure multiple domains

To configure for multi-domain support begins with the following 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:
   a. Set the `OEClientPrincipalFilter.key` property to a blank value.
   b. Set the value of the `OEClientPrincipalFilter.registryFile` property to the filename 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. (Optional) Configure `authz`, `accntinfo`, `roles`, and other properties.

The next step is to design the role name pattern that is used to designate an OpenEdge domain. Ideally, the name pattern has a unique prefix that distinguishes it from all other names. The name pattern may also employ a postfix to help establish uniqueness. For more information about designing a role name pattern, see Obtain a domain name from a role name on page 121.

Note: Because the `OEClientPrincipalFilter` bean uses Java regex functionality, create a role name pattern that is easy for Java regex to parse.

After you establish 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 5: Spring users.properties accounts

Role name	Spring Security role name	Java regex match pattern
`ROLE_OED:acme`	`ROLE_OED:acme`	`ROLE_OED:(.*)`

Table 6: 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 adds the prefix `ROLE_` to any LDAP group name it uses as a user account role, and it converts the LDAP group name to uppercase.

Finally, set the `domainRoleFilter` property of the `OEClientPrincipalFilter` bean to the value of the Java regex match pattern.

Note: If your Java regex match pattern contains a back slash (`\`), you may need to escape it with another backslash (`\\`).

About OpenEdge domains in LDAP configurations

Spring Security LDAP configuration details are almost always unique to each individual enterprise; however, the following gives 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 is 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 is 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 sees that `acmeuser1` is a member in the `ROLE_OED:acme`. The role supplies the domain name `acme` for creating a CLIENT-PRINCIPAL object's `domain-name` field.

For more information about LDAP, see About LDAP user accounts on page 98

Disable HTTP in production instances

As a security practice, disable HTTP in a production instance:

1. Run the Proenv utility.
2. Disable the HTTP feature for an instance with the PASMAN utility:

   ```
   pasman.{sh|bat} feature HTTP=off -I instance-name
   ```

3. Verify the feature statuses by running the PASMAN feature command:

   ```
   pasman.{sh|bat} feature -I instance-name
   ```

Use TLS in PAS for OpenEdge

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.

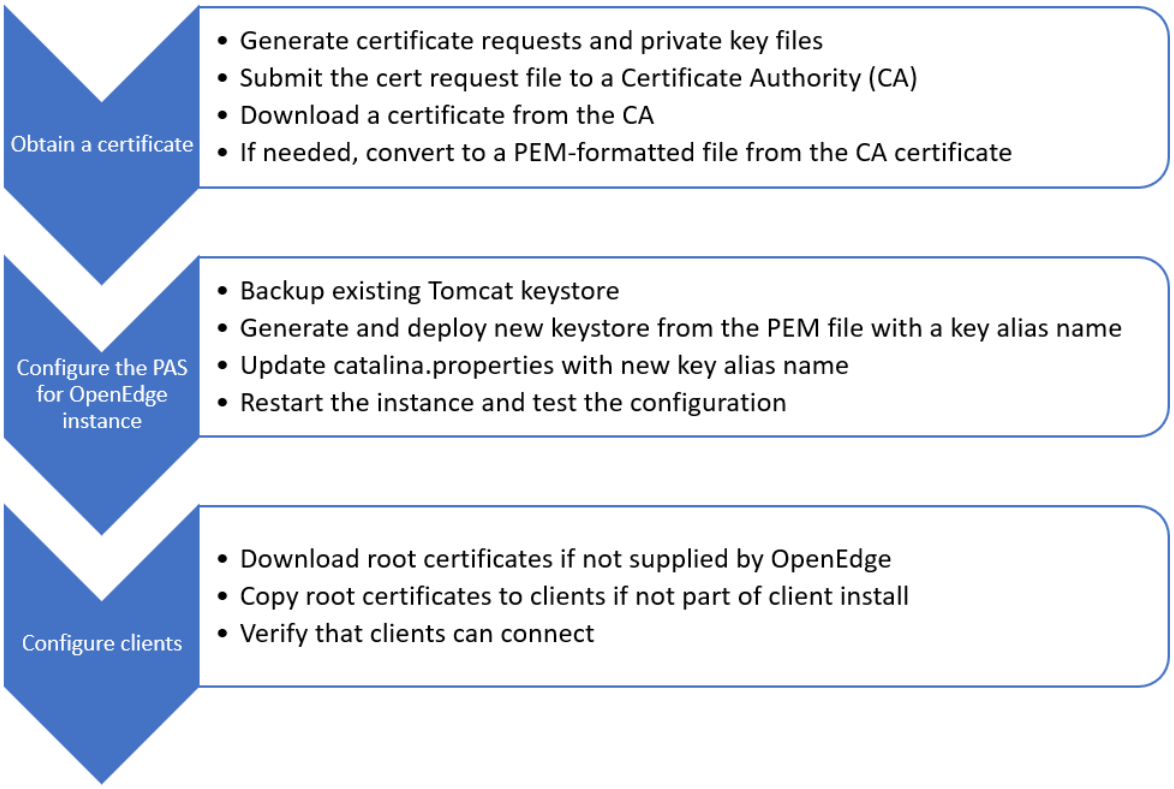
In PAS for OpenEdge and in Tomcat, HTTPS port support is provided by the Transport Layer Security (TLS) implementation of the Java Secure Socket Extension (JSSE).

TLS is a standard protocol that implements the encryption required for HTTPS communication.

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 TLS configuration in PAS for OpenEdge:



Important:

On a production instance, you *must* replace the default TLS server certificate shipped with PAS for OpenEdge with a fully qualified certificate from a certificate authority (CA).

Instance access through HTTPS is dependent on having an TLS server certificate. To support initial configuration and testing, Progress includes a self-signed server certificate in a Java keystore. (A keystore 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 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 TLS configuration:

- Obtain a certificate from a certificate authority on page 125
- Configure a PAS for OpenEdge instance for TLS on page 128
- Configure clients for TLS on page 127

Obtain a certificate from a certificate authority

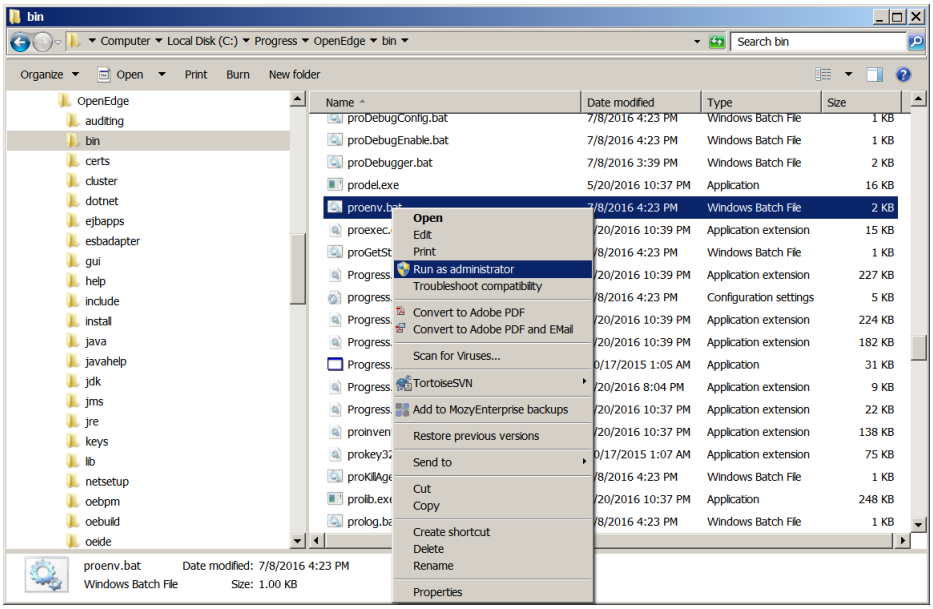
Generate a certificate request file that you can submit to a third-party certificate authority (CA) to obtain a TLS certificate, and import the certificate from the CA to a PAS for OpenEdge instance.

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

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

Note:

On Windows, your must start Proenv with the **Run as administrator** option.



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

```
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.

PKIUTIL prompts for the specific name of the system (server DNS name) and other information that is included in the certificate request. The information is incorporated into a distinguished name (DN). If a single period ('.') is entered, the field is left blank.

```
-----
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 get the signed TLS certificate.

   The CA returns the signed TLS certificate and the certificate is usually a file with either a `.crt` or a `.cer` extension.

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

5. Generate a Privacy-Enhanced Mail (`.pem`) formatted file from the private TLS certificate (`MyCertPriv.cer` in these examples) obtained from a CA.

   A PEM file is an encrypted file that contains keystore 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 *Manage OpenEdge Keys and Certificates*.

   a) In Proenv, change directory to the PAS for OpenEdge instance `/conf` directory:

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

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

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

   Note: `MyCert` is the stem filename of the PEM file that is 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.

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

Configure clients for 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 an instance:

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

   This directory should be where you copied the TLS certificates, as described in Obtain a certificate from a certificate authority on page 125.

2. Run certutil to import both the public and private 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 path name of the certificate file.
   - The key file alias names are eight hexadecimal characters in length and have a .0 (dot zero) file extension. However, keep in mind that your files do 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. For more information, see:

   - "Secure server connections to .NET Open Clients" in *.NET Open Clients*
   - "Secure server connections to Java Open Clients" in *Java Open Clients*

5. Test the client.

Configure a PAS for OpenEdge instance for TLS

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

Note: You 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 125).

To update a PAS for OpenEdge keystore:

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

```
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 *$CATALINA_HOME*/bin.

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

Note: Avoid using uppercase 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 uppercase 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 (the .pk1 file). See Obtain a certificate from a certificate authority on page 125.
4. Use the sslc command to verify what is now in the Tomcat keystore:

```
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:

```
# 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 lowercase letters.

6. Restart the instance and test.

   You can restart the instance using the TCMAN stop and start actions. After the instance restarts, test the connection and authentication configurations from a browser.

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

About client authentication

When you use TLS to secure data transfer between a client application and a server (in this case, PAS for OpenEdge), the server presents its certificate, digitally signed by a third-party certificate authority (CA), to the client application during the TLS handshake. Authenticating and establishing the identity of the server is part of the TLS handshake. This process is how the client validates its connection to the intended server.

Depending upon your business and security needs, you can also configure a client to authenticate its identity to the server. The configuration in which both the server and client authenticate is referred to as mutual authentication. Before you can use client authentication, the intended server must be configured to require a client certificate.

TLS client authentication is supported for SOAP, REST, WEB, and static file access through PAS for OpenEdge, and for ABL clients connecting to PAS for OpenEdge over APSV.

For more information about TLS in PAS for OpenEdge, see "Use TLS in PAS for OpenEdge" in *Manage Progress Application Server (PAS) for OpenEdge*.

How client authentication works

The following is the client authentication workflow:

1. The client sends a message to the server (a PAS for OpenEdge instance). The message contains details about the client identity and other security-related information.
2. The server responds with a message by sending its signed digital certificate and other security information required by the client.
3. If the authentication is successful, then the client sends its digital certificate to the server. Note the server must be configured to require the client certificate.
4. The server authenticates the client based on the digital certificate.
5. After the client and server are each authenticated, data transmission between the client and server begins.

Public and private keys, digital certificates, and trusted CAs

Private keys and digital certificates establish and verify identity and trust. TLS uses public key encryption for authentication. With public key encryption, a public key and a private key are generated. Data encrypted with the public key can only be decrypted using the corresponding private key. The public key is embedded in a digital certificate. A private key and a digital certificate provide identity for the ABL client.

The data embedded in a digital certificate is verified by a certificate authority (CA) and is digitally signed with the digital certificate of the CA. The digital certificate of the CA establishes trust.

Main steps for configuring client authentication

The main steps for configuring client authentication are:

1. Set up the client:
   a. Generate a public and private key pair.
   b. Generate a Certificate Signing Request (CSR) and submit it to a CA.
   c. Store the certificate that is signed and returned by the CA.
2. Configure the PAS for OpenEdge instance to require client certificates.

Configure ABL client authentication

When an ABL client initiates an HTTPS connection request with a PAS for OpenEdge instance, the client must include certain details in its request so that, in addition to the mandatory server authentication, the client authentication is performed during the TLS handshake.

Note: You can configure an ABL client for either TLS client or HTTP basic authentication (using `user ID` and `password`) but not both.

The steps in this procedure involve the use of the PKIUTIL command utility, which provides the operations to create and manage keystore entries for OpenEdge clients and PAS for OpenEdge instances. These operations include the ability to generate a Privacy-Enhanced Mail (PEM)-formatted file using the private certificate. A PEM file is an encrypted file that contains keystore information.

For more information about the syntax and usage of the PKIUTIL utility, see "pkiutil" in *Manage Keys and Certificates*.

To obtain a private key and digital certificate for an ABL client:

1. Create the private and public key pair for the ABL client:

   ```
   pkiutil -keysize 2048 -newreq client_alias
   ```

   In the preceding command, *`client_alias`* represents the name that corresponds to the keystore entry for the certificate and keys for the ABL client. When you execute this command, you are prompted for a password. You must use this password to access that keystore entry.

2. Submit the public key file (*`client_alias`*`.pk10`) to a certificate authority (CA) to request a signed certificate.

   The CA returns both your public key that the CA signed (your signed certificate), and the public certificate of the CA (the root certificate). These certificates have either a `.crt` or a `.cer` extension, for example, *`client_private`*`.cer` and *`client_public`*`.cer`. You can rename the certificates.

   Make sure that you save all certificates in a directory outside the OpenEdge installation directory. This ensures that the certificates are not deleted when you uninstall or re-install OpenEdge.

3. Import the client certificate (*`client_private`*`.cer`) into the client keystore:

   ```
   pkiutil -import client_alias client_private_certificate_dir\client_private.cer
   ```

   In the preceding command, *`client_private_certificate_dir`* represents the location where you stored the client certificate that you received from the CA.

   In this procedure, *`client_private`*`.pem` is generated.

   Note: The keystore in the OpenEdge installation directory is the client identity keystore.

   ---

4. When you are prompted, enter the password you used when you created the certificate and keys in Step 1. The PKIUTIL utility creates a certificate in a file, with the `.pem` extension, in the *`OpenEdge-install-dir`*`\keys` directory.

5. Use the following parameters in the `CONNECT( )` method for the `SERVER HANDLE` in the ABL client procedure:

Table 7: ABL client parameters for TLS client authentication

Parameter	Description
`-sslAuth` *`authentication_type`*	• Specifies if access to HTTPS requires TLS client authentication. • Set this parameter to `ssl` to enable client authentication for HTTPS access. • If the parameter is set to `basic`, then the `CONNECT()` method for the `SERVER HANDLE` does not perform client authentication. The default is `basic`.
`-sslKeyFile` *`filename`*	• Specifies the location of the client certificate (*`client_private`*`.pem`) file. • If you do not specify the absolute path of *`client_certificate_filename`*`.pem`, then the connection operation searches for the certificate file in the *`OpenEdge_install-dir`*`\keys` directory. • If `-sslAuth` is set to `ssl`, then the `-sslKeyFile` parameter must be specified. Otherwise, client authentication is not performed.
`-sslKeyPwd` *`password`*	• Specifies the password used to decrypt and use the private key of the client found in *`client_certificate_filename`*`.pem`. • The password can be in clear text or encoded using the `GENPASSWORD` utility.

Configure a PAS for OpenEdge instance for client authentication

Before you can use client authentication, you must complete a set of configuration tasks in PAS for OpenEdge:

1. Import the CA root certificate of the ABL client into the trust keystore used by the PAS for OpenEdge instance.
2. Enable TLS client authentication in the HTTPS port.
3. Configure Tomcat user accounts.
4. Set the **container** login model of the ABL web application.
5. Ensure that HTTPS support is enabled in your production instance.

To configure a PAS for OpenEdge instance for client authentication:

1. Import the CA root certificate of the ABL client into the trust keystore of the instance:

   ```
   keytool -importcert -alias CA_certificate-alias -keystore
   instance-name\conf\tomcat-certstore.jks -trustcacerts -file
   CA_certificate_filepath.cer -storepass storepass  -noprompt
   ```

   The CA root certificate is required to establish trust for the client certificate during an HTTPS connection.

   In the preceding command syntax:

   - *CA_certificate-alias*—The alias of the CA root certificate in the trust keystore
   - *instance-name*—The path of the PAS for OpenEdge instance directory on your machine
   - *CA_certificate_filepath*—The full directory path of the ABL client's CA root certificate that you are importing into the trust keystore of the instance
   - *storepass*—The trust keystore password

   Note: You must also import all required intermediate certificates into the trust keystore of the PAS for OpenEdge instance.

2. Configure the HTTPS port of the instance:

   a. Enable HTTPS client authentication for the instance by setting the value of the `psc.as.https.clientauth` property to `required`:

      ```
      tcman config psc.as.https.clientauth=required
      ```

      Note that this property has two additional values that can be specified: `none` and `optional`.

      If the property is set to:

      - `none`, which is the default value, then client authentication is disabled.
      - `optional`, then client authentication is performed if a client presents its certificate. But if the client does not present a certificate, then no authentication is performed, and a connection request is accepted.
      - `required`, then client authentication must be performed for all ABL web applications deployed to the instance.

3. Configure a user account (*username*, *password*, and *roles*) in the *instance-name*`\conf\tomcat-users.xml` file for each client that must use client authentication.

   When a client is successfully authenticated, Tomcat validates the client against the corresponding user account configured in the `tomcat-users.xml` file. Tomcat then grants or denies the client access to the instance according to the role that is defined for the user. The user account consists of comma-separated values, such as country (`C`), state (`ST`), organization (`O`), organizational unit (`OU`), and domain name server (`CN`). Note that you enter these details when you create your client certificate.

The following is an example user account added to the `tomcat-users.xml` file:

```
<user username="C=US,ST=client,O=Progress,OU=Openedge,CN=www.progress.com"
password="password" roles="ROLE_PSCAdmin,ROLE_PSCOper,ROLE_PSCUser" />
```

Note:

- Use the `pkiutil -list` command on the ABL client to retrieve details about the client username.
- Username details are displayed in the opposite order in the command window, starting with `CN` and ending with `C`. When you add the details to `tomcat-users.xml`, enter them in the following order: `C`, `ST`, `O`, `OU`, and `CN`.
- The user ID in the CLIENT-PRINCIPAL object is set to the Distinguished Name (`DN`) in the client certificate. The following is an example of the CLIENT-PRINCIPAL user ID:

  ```
  "CN=www.progress.com,OU=Openedge,O=Progress,ST=client,C=US"
  ```

4. Configure your ABL web application:

Configure the ABL web application security

The Spring Security framework, which is an integral part of PAS for OpenEdge, is adaptable to a wide variety of authentication models and architecture. To configure security for your ABL web application, you must enable Spring Security in the PAS for OpenEdge instance:

1. Open the *instance-name*`\webapps\`*webapp-name*`\WEB-INF\oeablSecurity.properties` file.
2. Enable Spring Security in the instance by setting the `apsv.security.enable` property to `container`.
3. Enable HTTPS client authentication in the `<login-config>` element in the *instance-name*`\webapps\`*webapp-name*`\WEB-INF\web.xml` deployment descriptor file:

```
<login-config>
 <auth-method>CLIENT-CERT</auth-method>
 <realm-name>OpenEdge</realm-name>
</login-config>
```

For ease of configuration, the `web.xml-clientcert` deployment descriptor file is provided in the same directory as the `web.xml` deployment descriptor file. The `web.xml-clientcert` file includes the required login configuration and security constraints. Make a backup copy of the `web.xml` file, and then rename the `web.xml-clientcert` file to `web.xml`.

By default, the security constraint defined in the `web.xml-clientcert` deployment descriptor file applies to all ABL web application transports. Customize the `<url-pattern>` element inside the `<security-constraint>` element according to your business needs.

The following is an example <security-constraint> element definition that restricts container security to the APSV transport:

```
<security-constraint>
        <display-name>PASOE oeabl Security Constraint</display-name>
        <web-resource-collection>
            <web-resource-name>Protected Area</web-resource-name>
            <url-pattern>/apsv/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>ROLE_PSCUser</role-name>
        </auth-constraint>
    </security-constraint>
```

4. To use **TLS Client authentication** for Webapp using APSV transport, perform the following:
   a. Open the `instance-name\webapps\webapp-name\WEB-INF\oeablSecurity.properties` file.
   b. Set `client.login.model=container`.
   c. Take a back-up of `instance-name\webapps\webapp-name\WEB-INF\oeablSecurity.csv` file.
   d. Open the `instance-name\webapps\webapp-name\WEB-INF\oeablSecurity.csv` file.
   e. In the **"APSV Container Login model Configuration"** section, uncomment the following line:

      `"/apsv/**","OPTIONS","hasAnyRole('ROLE_PSCUser','ROLE_ANONYMOUS')"`.

      And, comment the following:

      `"/apsv/**","OPTIONS","hasAnyRole('ROLE_PSCUser')"`.

   f. Then save and close the file.

Configure Spring Security

PAS or OpenEdge employs the Spring Security framework to provide enterprise-level application security for:

- **Authentication**—Verifies that username and password combinations are valid.
- **Authorization**—Enforces access control. User accounts are granted permissions according to their assigned roles.

Spring Security is a Java-based framework provided in the `$DLC/common/lib` directory of the OpenEdge install.

Implement a Spring Security model by supplying a reference to its configuration template in the `web.xml` file of a web application.

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.

About single sign-on support

Single sign-on (SSO) in OpenEdge refers to the creation of a sealed security token called the CLIENT-PRINCIPAL that represents a user identity within a security domain. The CLIENT-PRINCIPAL is created after Spring Security successfully completes authentication and authorization. The CLIENT-PRINCIPAL retains the user credentials and remains available within a web application, eliminating the need for a user to log in again.

In PAS for OpenEdge, basic Spring Security token generation is extended to create an OpenEdge CLIENT-PRINCIPAL object. Creation of the CLIENT-PRINCIPAL 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.

for more information about the properties and values for the `ClientPrincipal` and for SSO, See the `oeablSecurity.properties.README` file.

Note: In PAS for OpenEdge, SSO is available for client access through the APSV and REST transports but not for the SOAP transport.

About 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. These cross-domain requests are otherwise forbidden by the default, same-origin security policy of a web browser. The CORS standard defines how 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 is granted.

The CORS standard was designed primarily to support JavaScript clients; however, it may be applicable to other clients.

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. If the preflight request is approved, then 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, verify that your browser supports the CORS standard.

You must manage security constraints (for example, roles) in the `oeablSecurity.properties` file of the web application. For more information, see the *OECORSFilter bean* of the `oeablSecurity.properties.README`.

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`, where all of the primary CORS functionality is located. OpenEdge has integrated a 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 about 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.

Add CORS support

To add CORS support with a PAS for OpenEdge instance:

1. Open the `WEB-INF/oeablSecurity.properties` file of your web application.
2. Find the `OECORSFilter` bean.
3. Set `OECORSFilter` properties.

   For information about `OECORSFilter` properties, see the `WEB-INF/oeablSecurity.properties.README` file.

4. After saving your changes, restart the PAS for OpenEdge instance.

About managing 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 memory realm, 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 Role-based user authentication.

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.

About user and file permissions

The UNIX installation procedure of a production version of PAS for OpenEdge sets user and file permissions to limit who can access, manage, and configure the server.

Immediately after installation, all permissions for the PAS for OpenEdge base configuration (`$CATALINA_HOME`, which is set to `$DLC/servers/pasoe`) is owned by `ROOT` and is only accessible and executable by `ROOT` users and groups. Subsequently, you can set up your own groups with user accounts that have permission to run the `CATALINA_HOME` files. When you create a new instance from `CATALINA_HOME` while logged in with one of these new user accounts, you have access to the files on the new instance.

On Windows, no user or group access control lists (ACLs) are set during installation. Manually set permissions on the `%DLC%\servers\pasoe` directories, sub-directories and files to prevent unrestricted access.

About 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`. 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 the existing Tomcat roles they map to, if any:

Table 8: 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.

PAS for OpenEdge role	Includes Tomcat Manager and Host Manager roles	Description
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:

Table 9: PAS for OpenEdge default user accounts

Username	Password	Roles
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.

About remote access filters

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 code 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.

Authenticate using OpenEdge Authentication Gateway as a security token service

The OpenEdge Authentication Gateway can be configured as a Security Token Service (STS) for OpenEdge applications hosted on a PAS for OpenEdge instance.

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 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, see “Configure the OpenEdge Authentication Gateway” in *Learn about the OpenEdge Authentication Gateway*.

Authenticate using OAuth2 and JWT

Open Authorization (OAuth) 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 made up of components from multiple sources that must share a common user identity, OAuth2 and JSON Web Tokens (JWT) provide a way 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 topic 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 about configuring and troubleshooting.

Topics include:

- About OAuth2 on page 141
- About OAuth2 security considerations on page 143
- About OAuth2 tokens on page 143
- About JSON Web Tokens on page 145
- About support for OAuth2 and JWT in PAS for OpenEdge on page 148
- Configure a PAS for OpenEdge web application as an OAuth2 Resource Server on page 150
- Debug logging for OAuth2 on page 158

About OAuth2

This topic introduces core OAuth2 concepts and terms. Watch the following video to learn about OAuth2 support in PAS for OpenEdge.

Use OAuth2 with the Progress Application Server for OpenEdge.

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 a data owner, you 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

Actor	Description	OpenEdge Examples
Resource server	An abstraction that refers generally to any system or component that stores resource data and returns it to a client that is authorized to receive it. For example, 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, then it issues access tokens to a client on behalf of the resource owner.	N/A (third-party products)
User agent	A web browser, or some other type of framework that hosts client applications.	Browsers, mobile devices, Node.js, and so on
Authentication server	A server, undefined in the standard, that is used by an authorization server to authenticate the resource owner's identity. After the resource owner's identity is authenticated, the authorization server can continue its authorization process for issuing an access token.	N/A (third-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 through a user agent. The resource owner authenticates to an authorization server that 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 that 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 that 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 that 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.

About 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, and scanned, implementations
- Ensuring that every HTTP message travels through 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 to access data

About 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 resource server that uses the token's value to grant or deny a client's access to resource owner data.

Note: The server that produces the OAuth2 token is a third-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 (also known as an ID token, or identity token)
- Random access token
- Refresh token
- Bearer token

Authorization tokenss

An *authorization token* is a random data value issued by an authentication server to a client that then sends it to an authorization server and exchanges it for an access token (self-contained or random).

Self-contained access tokens

An OAuth2 *self-contained access token* is a Base64-encoded data value that holds the resource owner's identity and granted access rights (also known as scope). This form of access token can be used directly by a resource server's authorization process to grant or 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, but the OAuth2 standard does not specify the token's actual format. For more information, see About JSON Web Tokens on page 145.

Self-contained access tokens are commonly issued by enterprise authentication services to clients that can use the token in multiple resource servers. As explained in About OAuth2 on page 141, a resource server is an abstraction that refers generally to any system or component that stores resource data and returns it to a client that is authorized to receive it. Authorization is determined by the access token that is sent by the client. It is the responsibility of the resource server to limit the authentication services from which it accepts access tokens by validating the token before it may be used.

The general guideline states that an authorization server can issue a self-contained access token directly to trusted clients that send it to the resource server, in each service request, as a bearer token. The resource server's authorization process validates the self-contained access token's authenticity, and then uses the token's identity, or access right values, to grant or reject access and add to the service's audit trail.

Some authorization servers 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 logs in to the issuing authorization server and exchanges 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.

The method that 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 grants access to resource owner data.

Random access tokens

An OAuth2 *random access token* is a unique Base64-encoded random data value that is typically 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, or a resource owner's identity, or granted data rights, a resource server cannot use the token to grant access to a resource owner's data. Before an exchange of 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 tokens. The trust relationship can be established in several ways, and is not specified as part of the OAuth2 specification. Examples of a trusted relationship between an authorization server and resource server are 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 can (optionally) be issued by an authorization server in conjunction with a random or self-contained access token. To minimize the time a random or self-contained access token can be used after being stolen by a third party, an access token's expiration times can be kept very short. To not overly burden the resource owner with continually re-authenticating to obtain a new non-expired access token, an authorization server can be configured to issue a refresh token, which the client application can 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 typically takes place in the client when the resource server returns an expired token status. The resource server detects access token expiration only when an expiration value is present in the token and returns only a token expired error to the client. The resource server does not play a role in the client obtaining a new access token. When the client detects an expired token response from the resource server, enters into a refresh flow with the authorization server to obtain a new access token.

Bearer tokens

A *bearer token* refers to any form of identity token that is sent by a client to a resource server using an HTTP authorization header that identifies the authorization scheme as a 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"

About JSON Web Tokens

A JSON Web Token (JWT) is commonly used as the physical structure for an OAuth2 token, even though a JWT is not part of the OAuth2 standard. A JWT holds a resource owner's identity, client identity, issuing and expiration timestamps, and scopes used by a resource server's authorization process. Other authorization server-defined claims (also known as JSON fields) can 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 ('.').

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 and expiration timestamps, scope, client ID, and other values
Signature	Contains a binary digital signature of the header and 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 authorization server that issued the JWT token.	Yes	Yes
sub	The user ID. For OAuth2, it is the resource owner's ID.	Yes	Yes

aud	Can be a string value holding a single recipient identifier (also known as resource server ID) that limits the token's access to a single resource server. Can also be an array of strings with each array element holding a single recipient identifier, which then allows the token to access multiple resource servers.	Yes	Yes
client_id	This identifies the unique client ID registered by a client application with an authorization server, and is used by a resource server to determine whether the issued JWT can be used to access resource owner data.	No	Yes
jti	A unique identifier for this token that can 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 can not be used by a resource server.	optional	optional
nbf	The JWT validation date, before which it can 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 that this token was issued in (typically the bearer value).	optional	optional
<other>	Any other authorization server-provided claims.	optional	optional

A JWT's header can 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. For more information, see the table of JWS signature names.	Yes
typ	The type of token data format. If not specified, it often defaults 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

About support for OAuth2 and JWT in PAS for OpenEdge

PAS for OpenEdge support for OAuth2 allows your ABL business application or data service to act as a resource server that accepts either a OAuth2 self-contained access token (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-PRINCIPAL object that is usable in your ABL business application for setting the user of an OpenEdge database connection (and therefore tenancy and audit trail). Only those two types of tokens contain sufficient identity information to create a CLIENT-PRINCIPAL object.

Note: The OAuth2 random access token type is not supported because OpenEdge does not embed or interoperate with an external authorization server that is capable of sharing an OAuth2 self-contained access token with PAS for OpenEdge

Main steps for the HTTP request process

As an OAuth2 resource server, PAS for OpenEdge does the following on each HTTP request:

1. Obtains the authorization header's bearer token
2. Validates the bearer token type as either a JWT or OAuth2 self-contained access token (also known as a form of JWT)
3. Validates the JWT's signature using the JWT header's algorithm type and the PAS for OpenEdge configured encryption key
4. Validates the JWT's required payload claims
5. If the bearer token type is a JWT or OAuth2 self-contained access token, validates the required payload claims
6. Authorizes the token's user to the PAS for OpenEdge data services using the granted JWT scope claim
7. Creates an equivalent OpenEdge CLIENT-PRINCIPAL object that is delivered to the ABL business application with each request

PAS for OpenEdge support for OAuth2 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-PRINCIPAL objects 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 gives 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, which compose the default values if they are not found in the OEABL web application's configuration. OpenEdge provides many default property values that do not require changing. However, not all properties can contain useful values and must be configured for each installation.

PAS for OpenEdge supports most OAuth2- or JWT-compliant authentication and authorization services. However, there are limitations. An OAuth2 access token or JWT must meet the following criteria to be usable by the OAuth2 support in PAS for OpenEdge:

1. The access token, or JWT, must be received in an authorization HTTP header with the authorization-scheme set to 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 claim that corresponds to a resource ID that is configured in the OEABL web application's security.
5. An OAuth2 access token must contain a client ID claim (`client_id`) that indicates that it was issued to a client type that the OEABL web application is configured to support.
6. The token's signature algorithm must be one of the JWS, HMAC, or RSA types.
7. The encryption key to validate the token's signature field must be available from the issuer of the token and configured.
8. If the token issuer adds an expiration claim (exp), then the token must pass the date-time expiration test.

The following topics provide details about configuring PAS for OpenEdge's OAuth2 support:

- OAuth2 configuration files on page 149
- Enable OAuth2 support on page 150
- Configure a PAS for OpenEdge web application as an OAuth2 Resource Server on page 150
- Configure OAuth2 or JWT Token Validation Services on page 150
- Configure JWT/OAuth2 self-contained access token conversion to an OpenEdge CLIENT-PRINCIPAL on page 156

Note: Do not proceed without first understanding basic OAuth2, JWT, and OpenEdge CLIENT-PRINCIPAL terms and functionality, as explained in the following topics:

- About OAuth2 on page 141
- About OAuth2 security considerations on page 143
- About OAuth2 tokens on page 143
- About JSON Web Tokens on page 145

OAuth2 configuration files

The OAuth2 configuration in PAS for OpenEdge involves these files:

- `oeablSecurity.properties`—Contains property settings for JWT and OAuth2 validation operations and CLIENT-PRINCIPAL object generation.

  Note:

  There is a hierarchy of `oeablSecurity.properties` files, which exist in the following locations:

  - *`oe_install_dir`*`/servers/pasoe/conf/`
  - *`instance-name`*`/conf/`
  - *`instance-name`*`/ablapps/`*`abl-app-name`*`/conf/`
  - *`instance-name`*`/`*`web-app-name`*`/WEB-INF/`

  See Enable ABL application authentication for more information.

- `oeablSecurityJWT.csv`—Contains the resource server's URL access controls for JWT and 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)

Enable 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 that 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, then a second OpenEdge web application should be deployed and mapped to the same ABL application, where the alternate user login can 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.

Configure a PAS for OpenEdge web application as an OAuth2 Resource Server

Configuring a PAS for OpenEdge web application as an OAuth2 resource server requires the following:

- Configure OAuth2 or JWT Token Validation Services on page 150
- Configure JWT signature field validation on page 151
- Configure JWT payload field validation on page 153
- Configure self-contained access token validation on page 154
- Authorize access to URLs and HTTP Methods on page 154
- Configure JWT/OAuth2 self-contained access token conversion to an OpenEdge CLIENT-PRINCIPAL on page 156

Configure OAuth2 or JWT Token Validation Services

Set 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 varies depending upon 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, then its signature, and required JWT payload claims (aud and exp), are validated before they are passed to the URL authorization process. Using JWT tokens has more risk because 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 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.

Configure OAuth2 sessions

An OAuth2 resource server hosts a stateless REST API for its clients. As a stateless REST API, it does 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 }
```

Configure the WWW-Authenticate realm name

An OAuth2 resource server accepts access tokens using the HTTP authorization header. If the resource server's validation of the access token fails, then 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 the following property value in `oeablSecurity.properties`:

```
oauth2.resSvc.realmName=oeoauth
```

Configure 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 any of these checks fails, then 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 be performed by setting the following property in the `oeablSecurity.properties` file:

```
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: Because 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 is selected by the JWT header's `kid` claim

Configure 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 keystore is maintained by using the Java `keytool` utility. See Oracle documentation 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 used when you store the JWT RSA signature's public-key certificate that you obtained from the authorization server's operators.

Configure a PEM-encoded RSA public key

All files holding PEM-encoded RSA public keys must be in the form *file-name*.pem. You must configure the file system directory path (which acts as a keystore) and the alias for finding 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 key pairs can be generated for testing by 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 contains a single RSA public key and can be identified by an alias. When a JWT is received and its header contains a kid claim, the kid value is used as the alias that is used to look up a JWK object. If a JWT's header does not contain a kid claim, or a JWK cannot be resolved by alias, the request is rejected.

```
jwtToken.keystore.type=jwk
jwtToken.jwkKeyStoreURL=URL of where JWK RSA public keys will be loaded from
```

Configure JWT payload field validation

The following properties are used to validate a JWT self-contained access token's payload fields.

Validate the resource owner's ID claim

A JWT has a recommended field name (`sub`) to hold the authenticated user's ID, but can be configured differently by the issuing authorization server. See the JWT issuer's documentation for the field name that contains the authenticated user's ID:

```
jwtToken.usernameField=sub
```

Configure 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 inserts a payload audience (`aud`) claim, which is 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
```

Configure self-contained access token validation

Note the following the properties used to validate the additional claims that are 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 must 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.

Authorize 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. Although 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 to define the operations that a resource owner has authorized the resource server to perform for that client.

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`. This file performs the same basic function as the traditional RBA configuration file, `oeablSecurity.csv`.

The format of the `oeablSecurityJWT.csv` file has the same three double-quote fields that hold the URI, HTTP-method, and access-control specification. The first two fields (URI and HTTP method) are the same as in the RBA's `oeablSecurity.csv` file. 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 is granted a *name*, and that *name* corresponds to granting access to some process, function, or data. 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. The same rules apply as with RBA: the resource server's configuration must use the *scope* names configured and used by the authorization server. In production environments, that information must be obtained by the administrator.

An OAuth2 access-control expression has the following format:

```
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'...)`	Checks if OAuth2 client has one of the listed role names
`clientHasRole('name')`	Checks if the OAuth2 client as the specified role name
`denyOAuthClient( )`	Denies access to all OAuth2 clients, but allows other types of clients
`hasAnyScope('name' ...)`	Checks if the OAuth2 client has any named scope specified
`hasAnyScopeMatching( 'reg-ex'...)`	Checks if the OAuth2 client has one of the scopes matching a specified regex expression
`hasScope('name')`	Checks if the OAuth2 client has the named scope specified
`hasScopeMatching('reg-ex')`	Checks if the OAuth2 client has a named scope matching a specified regex expression
`isClient( )`	Checks if the request is from an OAuth2 client (and not another type of user)
`isOAuth( )`	Permits only OAuth2 type requests
`isUser( )`	Checks if the current request is from a user and not from an OAuth2 client

Each test returns true or false, and the entire expression must return true for the resource server to grant access.

Note:

In the default distribution of `oeablSecurityJWT.csv` the default scope names are:

- PSCUser
- PSCAdmin
- PSCDebug

Configure JWT/OAuth2 self-contained access token conversion to an OpenEdge CLIENT-PRINCIPAL

Similar to all Spring Security processes in PAS for OpenEdge, a successful authentication and authorization produces an OpenEdge CLIENT-PRINCIPAL object 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 Security and CLIENT-PRINCIPAL object 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 the JWT issuer's documentation to determine the field name that 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 using a CLIENT-PRINCIPAL object, then set the following option:

```
jwtToken.includeAllClaims={ true | false }
```

The creation and sealing of a CLIENT-PRINCIPAL object 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 need 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 can be filtered after 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 values 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 (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.

PAS for OpenEdge supports 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.

After the PAS for OpenEdge client application receives a SAML assertion from an identity provider, it sends the assertion to a PAS for OpenEdge web application, that then validates the assertion. If the validation is successful, then 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 the application.

About SAML

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 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 is structured and delivered.

The current version of the specification is **SAML 2.0**. The concepts described in this topic 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.

For more information 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, this is a SAML-based SSO workflow:

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, then 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 and 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 using 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 (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, and so on.
- 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, and so on).
- 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 Assertion Consumer Service (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 to pass information about the user's roles.

A SAML response can be signed or unsigned, and a SAML assertion can be signed or encrypted. The service provider and 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

Because 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, which is a unique identifier for the identity provider. This is typically an endpoint or URL.
- Supported HTTP bindings (such as POST or Redirect) 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, which is 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 also sent in the SAML request message.
- 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.

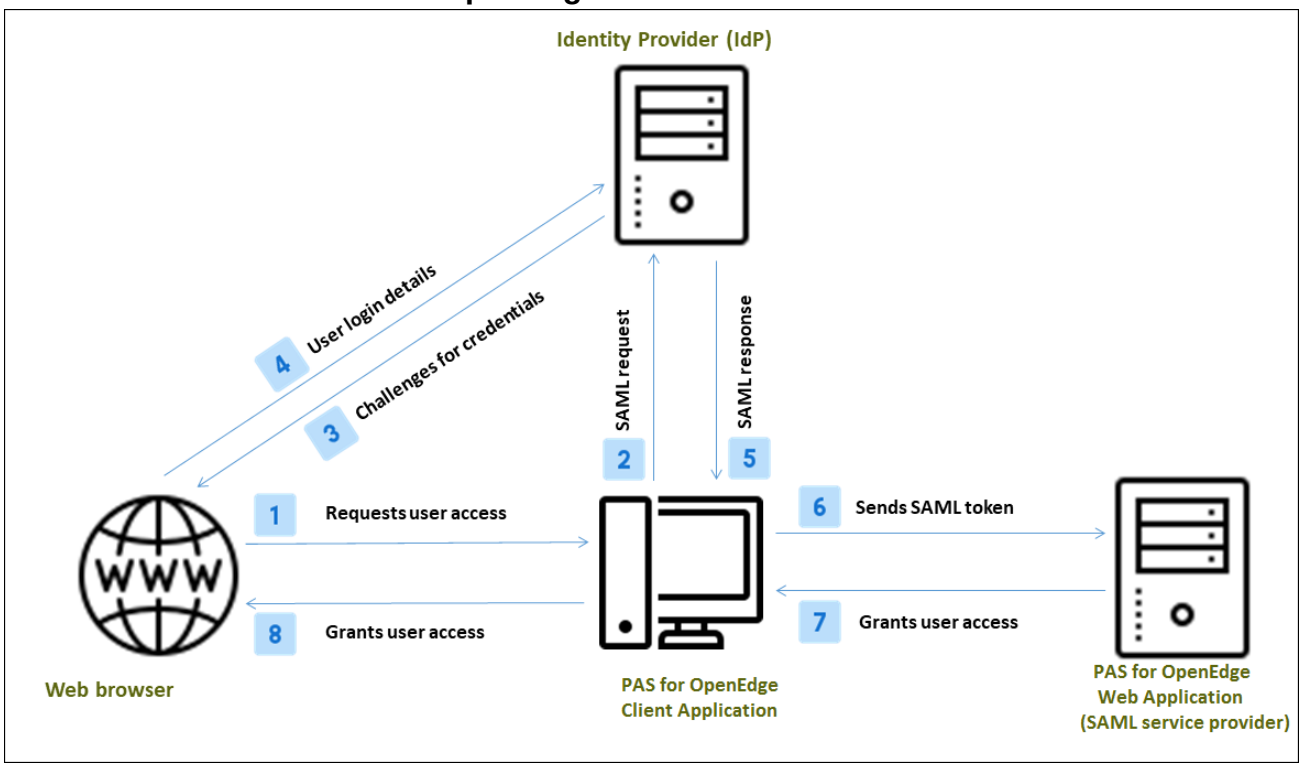
About support for SAML in PAS for OpenEdge

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 using 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 connects to a PAS for OpenEdge client application using a web browser 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 or signed, the PAS for OpenEdge web application uses the identity provider's public key (exposed using 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 expiration 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. **(Optional) ACS URL Validation**: 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 using 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 the 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 at sign ('@') 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, then 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, then you can modify a key-value property:

```
OEClientPrincipalFilter.properties.1.key=DOMAIN-DESCRIPTION
 OEClientPrincipalFilter.properties.1.value=OEApplication
```

Implement SAML in a PAS for OpenEdge web application

To implement SAML in PAS for OpenEdge, you must perform two tasks:

- Set up a PAS for OpenEdge client application.
- Configure SAML properties in a PAS for OpenEdge web application.

Set up a PAS for OpenEdge client application

Because 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 using 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.

Configure SAML properties in a PAS for OpenEdge web application

After you 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, download the metadata file of the client application and 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 name ID 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 name ID 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. For more information 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 values to "DEBUG":

```
<logger name="org.springframework.security.saml" level="DEBUG"/>
<logger name="org.opensaml" level="DEBUG"/>
<logger name=" PROTOCOL_MESSAGE " level="DEBUG"/>
```

Extend OpenEdge SSO to web applications

OpenEdge supports a single-sign on (SSO) framework in its application server and database product clients. PAS for OpenEdge extends the OpenEdge SSO framework to include support of mobile and browser-based clients (HTTP Clients) for OpenEdge ABL web applications. Providing tight integration with the existing SSO framework, SSO support for HTTP clients enhances ABL web application security, reduces the inconvenience of multiple logins, and relieves the need to write and maintain code that requires a high degree of technical security expertise.

The following topics describe how the existing OpenEdge SSO framework extends to web application clients, how that extended functionality is secured, and how the web application clients use those SSO services.

About PAS for OpenEdge SSO technologies

The PAS for OpenEdge extends OpenEdge SSO functionality into the Spring Security framework authentication and authorization process.

OpenEdge SSO support continues to authenticate user IDs and passwords, and to produce CLIENT-PRINCIPAL tokens for ABL application use. That basic OpenEdge functionality is extended with the capability to issue a simple type of SSO token to a client that it can pass to other PAS for OpenEdge web services that have SSO enabled. However, the token may not be refreshed (that is, its expiration extended) if the originating user login session terminated. If the SSO token is issued without the ability to be refreshed, then forcing a user to re-authenticate, the login session expiration limitation is moot.

The PAS for OpenEdge SSO support extends the original RESTful request support and adds a Spring Security filter to handle an upgraded client-server protocol. The PAS for OpenEdge SSO extensions provide both SSO token Producer and Consumer features for generating and consuming SSO tokens, respectively. Any one PAS for OpenEdge web service can be configured to produce, consume, or both. Additionally, a Spring Security configuration template (oeablSecurity-oesso.xml) allows a PAS for OpenEdge web service to only be accessed using a PAS for OpenEdge SSO token.

Because of the Spring Security framework's plug-in Authentication Provider architecture, PAS for OpenEdge SSO token generation can be configured to use any user-account provider supported by Spring Security or by an OpenEdge: properties file, such as LDAP; Active Directory; OERealm; SAML; OpenID; or CAS.

PAS for OpenEdge SSO tokens

The SSO access token in PAS for OpenEdge is a Base64-encoded and sealed CLIENT-PRINCIPAL. An optional Refresh token is a unique string value that is paired to one CLIENT-PRINCIPAL token.

A CLIENT-PRINCIPAL token minimally contains these fields:

- User ID and OpenEdge domain
- State SSO
- Expiration
- Roles (as granted by the Spring Security framework's existing behavior)
- Scope (which supplements Roles as a mechanism to further refine authorization rules. It limits clients with certain access tokens to certain web services, before Role URL authorization is tested.)

OpenEdge domain seal and validation support

PAS for OpenEdge SSO supports the sealing and validation of both single and multi-tenant CLIENT-PRINCIPAL tokens using the `OEClientPrincipalFilter` bean, which is configured in the `oeablSecurity.properties` file. The `OEClientPrincipalFilter` bean becomes the single point that manages all aspects of translating Spring tokens to CLIENT-PRINCIPAL tokens, the sealing of CLIENT-PRINCIPAL tokens, and the validation of CLIENT-PRINCIPAL tokens across all methods of direct-login and SSO.

All OpenEdge products that produce or consume CLIENT-PRINCIPAL tokens, including PAS for OpenEdge, are expected to define and use unique OpenEdge domain names, with each OpenEdge domain having a secret access code that is configured in each product that uses the OpenEdge domain. Each OpenEdge product is configurable to use some or all of the enterprise's defined OpenEdge domains, in effect adding a level of domain authorization to a Progress component. PAS for OpenEdge supports this architectural model

PAS for OpenEdge's web service SSO requires configuring a minimum of one domain or access code pair, and may support multiples by using a multi-domain registry (generated by the GENDOMREG utility). The ABL business application that receives an SSO token is expected to be configured with the same OpenEdge domain names and access codes. An SSO token must be validated using a domain configuration before it can be accepted as proof of the client's ability to use services and to be passed to the ABL business applications.

PAS for OpenEdge SSO configuration guide

Configuring PAS for OE SSO tokens is accomplished by updating the following files:

File path	Description
instance-name/conf/oeablSecurity.properties	Spring Security configuration defaults for all web applications
instance-name/webapps/*web-app-name*/WEB-INF/oeablSecurity.properties	Spring Security configuration settings for an individual web application
instance-name/webapps/*web-app-name*/WEB-INF/oeablSecurity.csv	URL access controls (Spring Security intercept-url settings) for individual web applications

Note: The `oeablSecurity.properties` files are where you configure the `OEClientPrincipalFilter` bean which manages all aspects of translating Spring tokens to CLIENT-PRINCIPAL tokens, the sealing of CLIENT-PRINCIPAL tokens, and the validation of CLIENT-PRINCIPAL tokens across all methods of direct login and SSO.

There are two SSO configurations, one for web applications that produce SSO tokens and one for web applications that consume SSO tokens.

Table 10: Overview of SSO Producer Configuration

Configure CLIENT-PRINCIPAL creation	• Add single or multi domain and access codes • Include Spring Security Authentication Provider granted roles • (Optional) Static Spring Security Roles for authorization to URLs

Configure SSO token creation	• Enable SSO token creation
	• Optionally change initial expiration time from 3600 seconds
	• Optionally enable SSO Token Refresh operations
	  • Optionally change refresh delta time of 3600 seconds
	  • Optionally define a SSO Token *scope* to filter which PAS for OpenEdge services are allowed to accept a SSO token generated by this service
	  • Optionally configure error level detail returned to the client
	  • Optionally allow HTTP messages instead of the required HTTPS

Note: Because of the security risks, PAS for OpenEdge web applications should not produce SSO tokens unless there are deployed clients capable of using the SSO that is produced. Therefore, the default setting for authentication and generation of native OpenEdge SSO tokens is disabled. In most cases, you can simply enable authentication or generation, or both.

Table 11: Overview of SSO Consumer Configuration

Configure CLIENT-PRINCIPAL validation	Add single or multi domain and access codes
Configure SSO Token use and validation	• Enable accepting SSO tokens for access to service URLs • Optionally configure error level detail returned to the client • Optionally allow HTTP messages instead of the required HTTPS

Configure the validation and use of native OpenEdge SSO token in client requests

The following table is a list of properties in `oeablSecurity.properties` that control if and how OpenEdge SSO tokens can be used to gain access to the data services.

Property name	Data type	Default	Value range	Description
`OESSO.error.detail`	integer	0	0 (none) 1 (terse) 2 (debug)	Controls the amount of error detail returned to a client for all SSO operations. The default (0) meets security best practices in returning little of value a hacker can make use of. But it does not supply an administrator or end-user with useful information for problem solving. Higher levels provide more information to administrators for problem resolution, but can also provide information usable by a hacker to attack your application. This property is used to set `OESSOFilter.authErrorDetail`
`OESSO.require.https`	boolean	`true`	`true`|`false`	When `true` controls the requirement for all SSO operations to require a client request to be made using the HTTPS URL scheme. When set to `false` allows HTTP (not recommended for operating a secure web application). This property is used to set the property `OESSOFilter.authSecurity`
`OESSOFilter.authPolicy`	string	disabled	—	See the following section, *OE SSO Token Consumer Policies*.
`OESSOFilter.authmanager`	string	string	—	Control which Authentication Manager is used to validate the OECP SSO token passed by the HTTP client. This property is mapped to the `http.all.authmanager` property and should only be changed when the OECP SSO Authentication-Manager must be different than the one used in the OECP SSO provider. See the `http.all.authmanager` property for valid settings.

Property name	Data type	Default	Value range	Description
`OESSOFilter.authScheme`	string	OECP	valid string	The HTTP Authorization header's authentication scheme field name that identifies the presence of an OECP SSO token value **Caution:** OpenEdge recommends that you *do not* change this value. Format: `Authorization: auth-scheme base64-sso-token-value`
`OESSOFilter.authClientType`	string	*	`regex`	Adds the ability to require the HTTP request's User-Agent: header to contain a specific value as defined by a Java RegEx pattern. The default "*" value disables User-Agent: header checking
`OESSOFilter.authErrorDetail`	int	---	---	Mapped to the property `OESSO.error.detail`. Explicitly defines a value if the error detail for consuming SSO tokens should be different from other SSO operations.
`OESSOFilter.authSecurity`	boolean	---	---	Mapped to the property `OESSO.require.https` Explicitly define a value if the requirement to use HTTPS should be different from the other SSO operations.
`OESSOTokenManager.ssoAllowScope`	string	*blank*	blank or list	When non-blank this property is used to control which OpenEdge SSO tokens may be used by this web application. If the OpenEdge SSO token passed by a client does has not been granted one of the scope values in this list, the token is rejected and the request fails. See the OpenEdge SSO token provider property `OESSOTokenManager.ssoGrantScope` for additional information.

OpenEdge SSO Token Consumer Policies

Policy Name	Description
`disabled`	The web application will not look for, or handle OpenEdge SSO tokens. If a native SSO token is passed in an Authorization header by a client it will be ignored

Policy Name	Description
`sufficient`	The web application will look for an HTTP Authorization header containing an authentication-scheme specified by the `OESSOFilter.authScheme` property (OECP) and a native SSO token. • If no HTTP Authorization header is present the authentication process continues with the next configured filter, which allows other SSO filters to operate. • If an HTTP Authorization header is present, but does not contain an authentication-scheme field matching the `OESSOFilter.authScheme` property, the authentication process continues with the next configured filter, which allows other SSO filters to operate. • If an HTTP Authorization header is present,and contains an authentication-scheme field matching the `OESSOFilter.authScheme` property, it will validate the contained OpenEdge SSO token value. If a validation error occurs, a 401 error will be returned to the client and no other SSO filters will be invoked. If validation is successful the native token will be extracted, the remaining authentication filters will be skipped, and the native token's ROLEs will be used to perform URL authorization. This policy is best used in the OpenEdge SSO token producer and OpenEdge SSO token consumer web applications that also support other forms of Spring Security direct-logins to user accounts.
`required`	

Policy Name	Description
	The web application will look for an HTTP Authorization header containing an authentication-scheme specified by the `OESSOFilter.authScheme` property (OECP) and a native SSO token. A 401 error response will be returned to the client and no other SSO filters will be invoked if any of the following conditions fail: • No Authorization header is found • An Authorization header is found, but is blank • An Authorization header is found but its authentication-scheme does not match the OESSOFilter.authScheme property • An Authorization header is found, has an authentication-scheme that does not match the `OESSOFilter.authScheme` property • An Authorization header is found, has an authentication-scheme that does match the `OESSOFilter.authScheme` property but does not have a SSO token value • An Authorization header is found, has an authentication-scheme that does match the `OESSOFilter.authScheme` property but does not have a valid SSO token value If successful the native token will be extracted, the remaining authentication filters will be skipped, and the native token's ROLEs will be used to perform URL authorization. This policy is best used in the OpenEdge SSO token consumer web applications that do not support any other forms of Spring Security direct-login to user accounts.

Configure the generation of OpenEdge native SSO tokens

Only certain types of PAS for OpenEdge web application may be a source of native SSO tokens. External authentication systems that produce their own security tokens and are integrated into Spring Security's process are prohibited from having an SSO token generated by OpenEdge.

The generation of an OpenEdge native SSO token occurs after Spring Security's authentication process completes a successful user direct-login. This post processing operation generates an extended ClientPrincipal security token that is safe enough to function in a less secure client login context environment (where a client receives and exposes a security token to the network world). This processing is handled by the existing OpenEdge authentication success handling, which uses the OpenEdge Client-Principal handler to do the physical Client-Principal generation and sealing. The OpenEdge successful authentication handler has the following configuration properties for producing OpenEdge SSO tokens, and adds some optional properties to allow tailoring to individual installations:

The following table is a list of properties in `oeablSecurity.properties` that control OpenEdge token generation properties.

Property Name	Data Type	Default	Value Range	Description
`OESSO.require.https`	boolean	`true`	`true` | `false`	When `true`, controls the requirement for all SSO operations to require a client request to be made using the HTTPS URL scheme. Set to `false` to allow HTTP (not recommended for operating a secure web application). This property is used to set the `OEAuthnSuccessHandler.tokenSecure` property.
`OESSO.error.detail`	integer	`0`	0 (none) 1 (terse) 2 (debug)	Controls the amount of error detail returned to a client for all SSO operations. The default (0) meets security best practices in returning little of value a hacker can make use of. But it does not supply an administrator or end-user with useful information for problem solving. Higher levels provide more information to administrators for problem resolution, but can also provide information usable by a hacker to attack your application. This property is used to set the `OEAuthnSuccessHandler.tokenErrorDetail` property.
`OESSOTokenManager.tokenPolicy`	string	disabled	—	Controls the actions to take when producing and returning a native OpenEdge SSO Token to a client. See the following *OESSOTokenManager Policies* section for more information.

Property Name	Data Type	Default	Value Range	Description
OESSOTokenManager.ssoTokenURLOption	string	oesso	"oesso" ""	The URL query option name used by a client to request the server issue a OpenEdge SSO token if the OESSOTokenManager.tokenPolicy value is ifRequired. For example: https://mywebserver/auth.jsp?security_token=yes A yes option value requests a OpenEdge SSO token be created and returned to the client if the user's authentication is successful. Any other option value will not create or return an OpenEdge SSO token. Ignored if the OESSOTokenManager.tokenPolicy property value is disabled or always.
OESSOTokenManager.ssoTokenRefresh	boolean	true	true|false	Controls a client's ability to request that an expired OpenEdge SSSO be refreshed with a new expiration date. NOTE: This property's value is forced to false if the OESSOTokenManager.tokenPolicy property's value is disabled.
OESSOTokenManabger.ssoTokenExpires	integer	3600 (1 hour)	0 – *n* seconds	This property controls the number of seconds a newly created OpenEdge SSO token is valid before it expires. If the value is less than 1, no refresh token will be generated and returned
OESSOTokenManager.ssoGrantScope	string	""	string	This property can be used to control which OpenEdge web applications may use an OpenEdge SSO Token produced by the Token Manager. It is used when an ABL application has many web applications, but not all of them should accept any OpenEdge SSO token. The *string* a comma separated list of names that correspond to web applications that should accept the OpenEdge SSO token. When blank, no scope information will be included in the OpenEdge SSO token. When Refer to the OpenEdge SSO token consumer property OESSOTokenManager.ssoAllowScope.

Property Name	Data Type	Default	Value Range	Description
OEAuthnSuccessHandler.tokenErrorDetail	integer	0	0-3	Controls the amount of error detail returned to a client during the OpenEdge SSO token creation process. This property is normally set using the OESSO.error.detail property to keep error detail information level consistent across the different SSO process operations. The property's value may be explicitly set if the OpenEdge SSO token generation errors return a different amount of error information than other SSO operations.
OEAuthnSuccessHandler.tokenSecure	boolean	true	true|false	Controls the requirement for HTTPS requests from the client while authenticating the user and returning an OpenEdge SSO token. This property is normally set using the OESSO.require.https property to keep the requirements for using HTTPS consistent across the different SSO process operations.This property may be explicitly set independently if the requirement for HTTPS is different from other SSO operations.

OESSOTokenManager Policies

Policy Name	Description
disabled	The web application will not generate OpenEdge SSO tokens. This policy must be used in all web applications that do not support the Spring Security HTTP Form direct-login to user accounts.

Policy Name	Description
`ifRequired`	The web application will only produce an OpenEdge SSO token if the client application requests it by using adding a URL option during a successful direct-login operation. For more information refer to the `OESSOTokenManager.tokenURLOption` property This policy should be used only in OpenEdge SSO token producer web applications that support the Spring Security HTTP Form type of direct-login to user accounts
`always`	The web application will always produce a OpenEdge SSO token if the client completes a successful direct-login operation. This policy should be used only in OpenEdge SSO token producer web applications that support the Spring Security HTTP Form type of direct-login to user account. **Note:** This is an extremely dangerous policy setting and should only be used in cases where the web application's authentication process is protected by other controls such as client types, IP addresses, etc.

Configure refresh of OpenEdge native SSO tokens

To mitigate client-side security holes, poor client application code security, and man-in-the-middle attacks, it is essential to limit the window of time that the SSO tokens handled by clients can be used. Using small windows of time requires the ability to transparently refresh an SSO token with a new expiration time. The refresh of an SSO token requires that only the authenticated user has the knowledge that the SSO token shared with the SSO token issuer. This ensures that the SSO token issuer can validate that only the authenticated user can obtain a refreshed SSO token.

The OE SSO token refresh provides a client with the ability to refresh an expired OpenEdge Native SSO token. The implementation intercepts a web application relative-URI, validates the client's refresh token, and, if successful, re-issues a new OpenEdge Native SSO token with a given lifetime.

OE SSO Token Refresh Properties

Property Name	Data Type	Default	Value Range	Description
OESSO.require.https	boolean	true	true | false	true sets the requirement for all SSO operations to require a client request to be made by using the HTTPS URL scheme. false allows HTTP (not recommended for operating a secure web application). This property is used to set the property OESSORefreshFilter.refreshSecure
OESSO.error.detail	integer	1	0 (none) 1 (terse) 2 (debug)	Controls the amount of error detail returned to a client for all SSO operations. The default (0) meets security best practices in returning little of value a hacker can make use of. But it does not supply an administrator or end-user with useful information for problem solving. Higher levels provide more information to administrators for problem resolution, but can also provide information usable by a hacker to attack your application.
OESSORefreshFilter.refreshURL	string	/static/oe	viable relative URI	Sets the relative URI that a client will use to execute a refresh operation for an expired OE SSO token. The property's value may not be blank or in any of the following URI spaces: • /apsv • /soap • /rest • /web

Property Name	Data Type	Default	Value Range	Description
`OESSORefreshFilter.refreshURLOption`	string	refresh	{ `refresh` }	The URL query option name that contains the type of token operation requested by the client. For example: https://myservice/sts/auth/token?refresh
`OESSORefreshFilter.RefreshClientType`	string	`*` (every client type)	Java RegEx pattern	Adds the ability to require the HTTP request's User-Agent: header to contain a specific value as defined by a Java RegEx pattern. The default `"*"` value disables `User-Agent: header` checking.
`OESSOTokenManager.ssoRefreshDeltaTime`	integer	`3600` (1 hour)	`1` – *n* seconds	Controls the amount of time, in seconds, that a refreshed OE SSO token will remain valid before it expires and must be refreshed again.

Property Name	Data Type	Default	Value Range	Description
`OESSORefreshFilter.refreshErrorDetail`	integer	`0`	`0-2`	Controls the amount of error detail returned to a client during the OE SSO token refresh process. This property is normally set using the `OESSO.error.detail` property to keep error detail information level consistent across different SSO process operations. The property's value may be explicitly set if the OE SSO token refresh errors should return a different amount of error information than other SSO operations.
OESSORefreshFilter.refreshSecure	boolean	`true`	`true|false`	Used to control the requirement for HTTPS requests from the client while refreshing an OE SSO token. This property is normally set using the `OESSO.require.https` property to keep the requirements for using HTTPS consistent across the different SSO process operations.This property may be explicitly set independently if the requirement for HTTPS is different from other SSO operations.

PAS for OpenEdge guide to SSO token handling

This topic describes SSO authentication models and is for HTTP client application developers who want to use native OpenEdge SSO token functionality. This topic also contains information about the formation of HTTP requests and interpretation of HTTP responses.

HTTP FORM authentication model

The HTTP FORM authentication model provides user session support when the client uses application defined URL resources for login/logout operations. The client supplies the user's identification assertions (such as user ID and password) in a POST request's body and receives session information in the POST response's headers. For each subsequent HTTP request, the session information received as part of the login operation is passed as HTTP header information.

Applying SSO to this authentication model involves obtaining the native OpenEdge SSO token created and stored as part of the user login process, and passing that SSO token to other web applications that are configured to accept it.

Client request

```
POST web-app-url/static/auth/j_spring_security_check?OECP=yes
Content-Type: application/x-www-form-urlencoded
j_username=userid&j_password=pwd
```

Server response

```
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
set-cookie : JSESSIONID=user-session-reference

{ "token_type" : "oecp",
 "access_token" : "b64-oecp-sso-token",
 "refresh_token" : "oecp-ref-token",
 "expires_in" : int-seconds
}
```

Note: `refresh_token` may be blank, indicating that the `access_token` cannot be refreshed.

HTTP status codes

- **200** indicates successful server response.
- **400** indicates an SSO token generation failure.

  ```
  Content-Type: application/json;charset=UTF-8
  Cache-Control: no-store
  Pragma: no-cache

  "error_description" : "error-desc
  }

   token-error-code
  ```

  For more information about *token-error-code*, see SSO token error codes on page 184.

- **401** indicates user authentication failure for data services.

```
WWW-Authenticate : http-form realm info
```

- **500** indicates an internal server failure.

HTTP BASIC authentication model

The HTTP BASIC authentication model does not support user login sessions that can be used for maintaining the server context required for issuing or maintaining SSO tokens and their refresh tokens. Therefore, HTTP BASIC cannot produce SSO tokens.

SSO authentication model

The SSO authentication model for native OpenEdge SSO tokens follows the HTTP BASIC model, where a client's identity is authenticated on each HTTP request. The difference between the two is that HTTP BASIC sends user ID and password authentication on each request, and SSO sends an encoded SSO token in each request.

Client request

The HTTP 1.1 format for the request is:

```
"Authorization : " + token_type + " " + access_token
```

For OpenEdge, the client request is:

```
Authorization : oecp b64-oecp-sso-token
```

HTTP status codes

- **200** indicates success.
- **401** indicates an SSO authentication failure.

```
WWW-Authenticate : oecp error=401,
error_description=error-desc
token-error-code
```

For more information on *token-error-code*, see SSO token error codes on page 184.

SSO refresh model

All SSO tokens, including the native OpenEdge SSO token, must expire after some set interval. They can be refreshed either by:

- Performing a full direct login by the client

- Returning to the point where an SSO token was issued and requesting a new token with an extended expiration

Refreshing a native OpenEdge SSO token takes place using a defined URL (similar to the way HTTP form login is implemented). In this case, the client uses a POST request to a URL, and passes the refresh token it received with the last SSO token it obtained. If the refresh operation is successful, it returns a new SSO token and, optionally, a new refresh token.

Client request

```
POST web-app-url/static/auth/token?op=refresh
{ “token_type” : “oecp”,
 “refresh_token” : “oecp-ref-token”
}
```

Server response

```
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
 “token_type” : “oecp”,
 “access_token” : “b64-oecp-sso-token”,
 “refresh_token” : “oecp-ref-token”,
 “expires_in” : int-seconds
}
```

HTTP status codes

- **200** indicates successful server response.
- **401** indicates SSO token generation failure.

```
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
“error” : “401”,
“error_description” : “error-desc”
}

 token-error-code
```

For more information about *token-error-code*, see SSO token error codes on page 184.

SSO token error codes

The following table describes the SSO token-operation error codes.

Error Code	Description
internal_error	An internal error occurred while creating or signing a CLIENT-PRINCIPAL at user authentication time; no SSO token was produced.
expired_token	The client-submitted token expired, and access to the web application was denied.
rejected_token	The token value was rejected.
invalid_client	The token request was issued from an invalid client type.
invalid_token	The client-submitted SSO token failed validation, and access to the web application was denied.
corrupt_token	The token submitted to an SSO token is corrupt (could not validate seal), and access to the web application was denied.
invalid_request	The request containing information about the SSO token is incorrect or incomplete, and access to the web application was denied.
unsupported_token	The request contains an unsupported token type, and access to the web application was denied.

About support for server name indication

If your OpenEdge Authentication Gateway server is configured with multiple virtual hosts that are bound to a single IP address, then you can use a parameter called `-servername` to specify the hostname that you want to connect to in the STS Client and the STS URL utilities.

STS client utility (stsclientutil)

The `stsclientutil` utility is an administrator-level utility used in troubleshooting OpenEdge database connections to the Secure Token Server (STS) application. Specifically, `stsclientutil` addresses these tasks:

- Establishes the OpenEdge installation's ability to connect to a PAS for OpenEdge server and use its STS application (that is, *ping* the STS).
- Tests the ability of the OpenEdge installation to use an STS application for a direct user login operation for a specific OpenEdge domain (that is, *authenticate* the user).
- Tests the ability of the OpenEdge installation to use an STS application for an operating-system, user login SSO to a specific OpenEdge domain (that is, *exchange* the OS security token for an OpenEdge security token).

Command syntax

```
stsclientutil -cmd command -url sts-url
      [command-options]
```

`-cmd` *command*

Specifies the task to perform from one of the following:

- **`ping`**—Sends an empty message to an STS application to test HTTPS connections and STS key features.
- **`authenticate`**—Sends 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:

  ```
  -user user[@domain]
  ```

- **`exchange`**—Sends 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:

  ```
  -domain domain
  ```

Note: There is no default for *`command`*; one of the options must be specified.

`-url` *sts-url*

Specifies the URL of the STS application (secure PAS for OpenEdge 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 }`

Specifies 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*`...]`

Specifies 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`*

Specifies 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`*

Specifies 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`*

Specifies 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`*

Specifies 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`

Suppresses 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 TLS certificate during the TLS handshake instead of the Server URL's host.

The `stsclientutil` utility is a secured application that can be 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
```

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`*

Specifies the new URL for the Authentication Gateway to insert into the database configuration.

Specify the *`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 TLS certificate during the TLS handshake instead of the server URL's host.

`-folder` *`folder-name`*

Specifies 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`*]

Specifies one or more OpenEdge databases to connect to. *`db-options`* allows you to specify additional connection parameters.

[`-db` *`dbname`* [*`db-options`*] `...`]

Specifies additional databases and connection options.

`abl-startup-options`

Specifies additional ABL connection parameters.

The `stskeyutil update` utility writes the new URL of the gateway to the database.

5

Tune PAS for OpenEdge instances

A well-tuned PAS for OpenEdge instance balances the use of the OS, CPU, memory, and processes to service the largest number of concurrent client requests with the best response time. A poorly tuned PAS for OpenEdge instance can overload the server instance, multi-session agent, or OS which degrades client response times or causes server outages.

This content describes the parameters that can be adjusted to improve the performance of your instance and application in production.

See also:

- PAS for OpenEdge tuning recommendations
- Goals and common steps for tuning PAS for OpenEdge instances
- PAS for OpenEdge server-side tuning
- Tune OpenEdge web applications and MS-agents

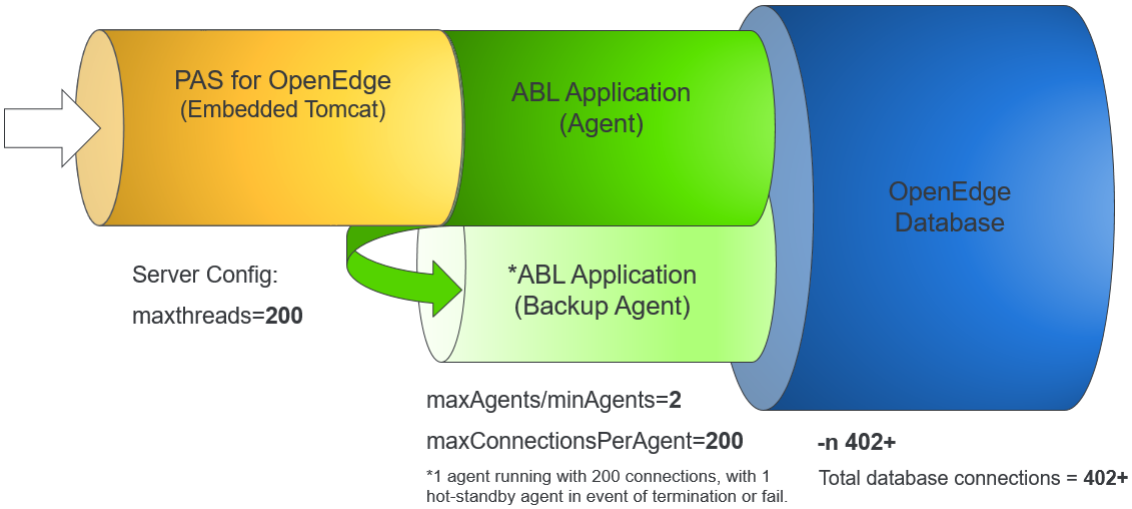
For details, see the following topics:

- PAS for OpenEdge tuning recommendations
- Goals and common steps for tuning PAS for OpenEdge instances
- PAS for OpenEdge server-side tuning
- Tune OpenEdge web applications and multi-session agents

PAS for OpenEdge tuning recommendations

Before you read a detailed account of PAS for OpenEdge tuning in the following topics, know that there are a few parameters that you can tune to affect the performance of your application immediately. If the parameters are set correctly, then they could improve the performance of your application in production.

The following figure shows a configuration in which neither the server nor ABL application agents throttle connections to the database because the PAS for OpenEdge instance and ABL applications were tuned:



The figure shows a PAS for OpenEdge instance that:

- Can run 200 concurrent client requests to a single stateless/state-free ABL application.
- Includes a hot standby, multi-session agent that enhances application availability. A hot standby is a redundant method in which one system runs simultaneously with an identical primary system.

This configuration can be mimicked in your environment or altered based on your needs. This example configuration does not show how to load balance multiple instances or applications. This topic explains how to tune a single instance, to get the best performance. Scaling and sizing considerations are discussed in the Scaling considerations in PAS for OpenEdge and Load balance PAS for OpenEdge instances.

Server-side tuning recommendations

To set a server configuration that can deliver the best application performance, change these parameters:

Server configuration properties	Description
`psc.as.executor.maxthreads=200`	The maximum number of client requests (request execution threads) that can be executed by the server before they are placed in a wait queue. The maximum applies to all deployed Java or ABL applications.
• `psc.as.http.maxqueuesize=100` • `psc.as.https.maxqueuesize=100`	The maximum number of client requests that can be queued, waiting for a request execution thread.

The server configuration properties can be set using the `PASMAN config` command. For more information, see Use PASMAN to configure an instance.

The sum of the properties (`maxthreads` and `maxqueuesize`) plus the size of the client request data affects the total process memory size of the PAS for OpenEdge instance process.

The `psc.as.executor.maxthreads` configuration property determines the limit for how many concurrent client requests the PAS for OpenEdge instance can execute. Because each client request executes in one of these threads, the value must be as large or larger than the total number of client requests by all deployed Java or ABL applications. PAS for OpenEdge, which runs an embedded version of Tomcat, does not load balance client requests by deployed application. Therefore, if the thread size is not large enough, you may find that client requests to random applications are queued and pause the application. The number of queued threads is set with the `psc.as.http.maxqueuesize` or `psc.as.https.maxqueuesize` configuration property. For more information about tuning the server configuration, see PAS for OpenEdge server-side tuning.

ABL web application tuning tips

To set an ABL web application configuration that can deliver the best application performance, change these parameters:

ABL application configuration	Description
`numInitialAgents=2`	Starts two multi-session agent OS processes: one handles all client requests and one remains idle in hot-standby mode.
• `minAgents=2` • `maxAgents=2`	Maintains two running multi-session agent processes at all times; if one agent stops, then a second is automatically started.
`maxConnectionsPerAgent=200`	Sets the maximum number of concurrent client requests handed by the PAS for OpenEdge server (set equal to `psc.as.executor.maxthreads`).
`maxABLSessionsPerAgent=200`	Sets the maximum number of ABL sessions handled by each multi-session agent (set equal to the `maxConnectionsPerAgent` in stateless/state-free ABL application architectures).

The ABL web application configuration properties can be set using the OEPROP utility, for example:

```
oeprop.[sh|bat] AppServer.SessMgr.numInitialAgents=2
oeprop.[sh|bat] AppServer.SessMgr.minAgents=2
oeprop.[sh|bat] AppServer.SessMgr.maxAgents=2
oeprop.[sh|bat] AppServer.SessMgr.maxConnectionsPerAgent=200
oeprop.[sh|bat] AppServer.SessMgr.maxABLSessionsPerAgent=200
```

This configuration runs the maximum client request load (`maxConnectionsPerAgent`) of 200 using one agent OS process. The second agent OS process stays in standby mode until the first agent OS process is stopped by the system administrator directive or the process fails. You do not have to manually create a second agent OS process if one terminates because PAS for OpenEdge automatically creates the second process.

The sample ABL web application configuration assumes that after load testing, a single agent OS process can run 200 client requests concurrently without exceeding the OS process, CPU, memory, or I/O limits. For more information about tuning ABL application configurations, see Tune OpenEdge web applications and MS-agents.

For information about machine sizing and to download a sample application that can be used for testing the tuning properties and size of the PAS for OpenEdge instance, see the PAS for OpenEdge ATM Benchmark Test and Machine Sizing Guide on Progress Community.

Goals and common steps for tuning PAS for OpenEdge instances

The overall goal for tuning a PAS for OpenEdge instance is to support an ABL application that meets your requirements for concurrent client requests, response times, and resource consumption. Achieving this goal is an iterative process that begins when you develop the ABL application and set the initial defaults for the target deployment environment. You then perform more refined tuning of the OS, CPU, memory, and file system.

The tuning process typically consists of the following steps:

1. Tuning the PAS for OpenEdge instances.
2. Tuning the OpenEdge application and any third-party web applications.
3. Tuning the OpenEdge multi-session agent.

The following is a general set of steps to tune the PAS for OpenEdge instance:

1. Deploy the OpenEdge web applications that make up your ABL application to the PAS for OpenEdge instance, increasing the JVM metaspace memory until all web applications start successfully.
2. Determine and set the maximum number of ABL sessions that a single multi-session agent can support based on your ABL application memory allocation of variables, r-code, and buffer space.
3. Calculate the maximum ABL session pool size where the maximum number of ABL sessions across all multi-session agents is based on the sum of:
   - The estimated maximum number of concurrently bound client application connections.
   - The estimated maximum number of concurrently executing inbound client requests.
4. Set the maximum number of multi-session agents as (*`session-pool-size / maximum sessions per multi-session agent`*) `+1`. The `+1` is for handling abnormal load spikes. Use the OEPROP utility to set the `AppServer.SessMgr.maxAgents` property.
5. Set the maximum number of local socket connections (`maxConnectionsPerAgent`) that a single multi-session agent can have. The goal is to maximize the number of concurrently running client requests that one multi-session agent can process. The CPU resources and the level of I/O switching of the individually executing ABL requests influence the value. This number must be less than or equal to the maximum number of ABL sessions for each agent. Use the OEPROP utility to set the `AppServer.SessMgr.maxConnectionsPerAgent` property.
6. Set the idle resource timeouts to lower the resource load on the server without causing multi-session agent or ABL session thrashing. These settings are directly related to how long it takes an ABL session to be started in your ABL application. Use the OEPROP utility to set the `AppServer.SessMgr.idleResourceTimeout` property.
7. Set the number of initial ABL sessions an multi-session agent starts when it is created. Keep this number to a minimum because it affects how long it takes for the PAS for OpenEdge instance to become available to handle the client load. The goal is to initialize enough ABL sessions to handle the initial client load, and then to gradually scale up (trading off the initial client response times) until the PAS for OpenEdge instance reaches full capacity. Use the OEPROP utility to set the `AppServer.Agent.numInitialSessions` property.

Keep in mind that the maximum capacity of a single PAS for OpenEdge instance is finite for any given combination of ABL application and OS hardware. When you reach a single PAS for OpenEdge instance limits within the current environment, and you have not met the performance requirements, you must either increase the OS hardware capacity, or run additional PAS for OpenEdge instances and load balance the instances.

For information about how to scale PAS for OpenEdge instances, see Scaling considerations in PAS for OpenEdge.

PAS for OpenEdge server-side tuning

Tuning the PAS for OpenEdge instance is an iterative process of monitoring instance behavior for a given client and web application configuration, changing configuration values, and then returning to the monitoring phase to see if performance is improved. System administrators can control the following attributes from a PAS for OpenEdge configuration:

- JVM stack, heap, and metaspace memory allocations
- JVM garbage collection cycles and overhead
- Number of HTTP client connections
- Size of the thread pool used to execute HTTP client requests
- Client connection timeout (elapsed time without an HTTP request)
- Queue size for holding HTTP client requests until a thread is available from the pool
- Socket buffer size
- HTTP compression for SOAP and REST clients
- Turning Tomcat optional features on or off
- Determining the web applications that will be deployed to the PAS for OpenEdge instances

JVM configuration parameters are located in the *instance-name*`/conf/jvm.properties` file as Java system properties. You manually edit this file to make changes.

PAS for OpenEdge configuration parameters are located in the *instance-name*`/conf/catalina.properties` file as Java system properties, where they are easier to manage from local scripts and remote administration tools. Those Java system property values are used as variable values in the *instance-name*`/conf/server.xml` file. Normally you do not edit the `server.xml` file because it may cause the server to not start. You can use the PASMAN command-line utility or a text editor to maintain the `/conf/catalina.properties` file.

Use Java tools, commercial tools, or open source products to monitor PAS for OpenEdge resources and operations.

JVM parameters

The most important aspect of tuning your PAS for OpenEdge instance is the JVM stack, heap, and metaspace memory allocations and garbage collection. The memory configuration ultimately determines how many and the type of web applications that can be deployed on the PAS for OpenEdge instance and the number of clients that the instance can support for those web applications.

PAS for OpenEdge process memory and garbage collection are a critical part of your tuning strategy. PAS for OpenEdge instances perform poorly when either the heap or metaspace memory regions run low on space. To keep memory available to the JVM, its garbage collector must run periodically to reclaim unused space. Each time the garbage collector runs, it stops all of the other JVM threads such as client HTTP request execution threads.

If the garbage collector runs too infrequently:

- Large amounts of unusable memory may accumulate.
- The collection process takes longer and results in spikes in client response times.

If the garbage collection process runs too frequently:

- You lose CPU cycles while checking the entire memory space.
- Client response times grow longer due to constant CPU drain by garbage collection cycles.

The goal is to ensure that PAS for OpenEdge has sufficient heap and metaspace memory allocated to handle the loading of web applications, web application memory allocations, and concurrent client requests to reduce garbage collection processing time.

The properties that influence memory consumption include:

- The number and type of web applications deployed
- The size of the HTTP request thread pool
- The number of concurrent requests and the data size of those requests

The following JVM parameters configured in the *instance-name*`/conf/jvm.properties` file are relevant in determining memory and garbage collection limits:

Property	Default	Description
`-Xms<`*size*`>`	`512m`	Initial Java heap size
`-Xmx<`*size*`>`	`1024m`	Maximum Java heap size
`-Xss<`*size*`>`	`1024k`	Java thread stack size
`-XX:NewSize`	`64m`	Initial space used for short duration objects and indirectly how often garbage collection runs
`-XX:MaxNewSize`	`64m`	Maximum space used for short duration objects
`-XX+DisableExplicitGC`		Disable explicit garbage collection
`-XX:MetaspaceSize`	`unlimited`	Initial Metaspace size.

Tuning tips

- Reduce the frequency of garbage collection by starting the JVM with a larger maximum heap memory space (`-Xmx`).
- Reduce repeated reallocation of heap memory by setting the initial heap memory space equal to the maximum amount (`-Xms == -Xmx`).
- Reduce repeated reallocation of metaspace memory by setting the initial size equal to the maximum size (`-XX:MetaspaceSize == -XX:MaxMetaspaceSize`).

- Carefully lower the JVM stack size (`-Xss`) to save process memory for heap and metaspace allocations.
- Increase metaspace memory when an instance stops due to an out-of-metaspace error.

Network connections

Tuning the PAS for OpenEdge client network connections involves controlling the HTTP client TCP/IP connections. Many of the network connection properties are defined as Java system properties in the *instance-name*`/conf/catalina.properties` file and have `psc.as.` name prefixes. These properties can be managed using the PASMAN command-line utility. Each named `psc.as`... Java system property in the `catalina.properties` file is related to an XML element or attribute in an Apache Tomcat *instance-name*`/conf/server.xml` file. You can find that relationship using the command `pasman.[bat|sh] help psc.as.`*xxxx*.

Note:

Progress recommends that you do not edit the *instance-name*`/conf/server.xml` file unless an Apache Tomcat configuration attribute is needed that is not supplied by a `psc.as` configuration property. If a new Apache Tomcat configuration attribute is required, Progress recommends that you add the attribute to the following locations:

- The `server.xml` file using a `psc.as.` property
- The `catalina.properties` file, which can be managed remotely using administration tools and automated scripts

---

PAS for OpenEdge client network connections are a server-level resource and exist for each open HTTP, HTTPS, and AJP13 port. You cannot control client network connections for each web application. Your primary goal is to ensure that PAS for OpenEdge has enough network connection capacity to handle the total client load for all deployed web applications including OpenEdge web applications and any additional external third-party web applications.

Tuning tips

- Coordinate the maximum number of client connections to be equal to or larger than the size of the PAS for OpenEdge thread pool and the number of queued HTTP client requests.
- Do not attempt to enable HTTP message compression for HTTP-connected OpenEdge clients by adding its MIME type to the `psc.as.compress.types` property because it will not work.
- The HTTP-connected OpenEdge clients use `HTTP POST` messages with a maximum size of 8 KB, so the maximum `psc.as.msg.maxpostsize` is not an issue. If you are using REST or SOAP clients with very large ProDataSet transfers, then this property setting may become important.

The secondary goal is to modify the HTTP message handling if the default settings do not allow the very large message or response data exchange required by some web applications.

HTTP connection properties

Set these HTTP connection message-handling properties to increase network throughput:

HTTP property	Default	Description
psc.as.HTTP.connectiontimeout	20000	The maximum time in milliseconds between a TCP connection and the appearance of an HTTP or HTTPS message.

HTTP property	Default	Description
psc.as.HTTP.maxconnections	-1	Maximum client connections on the HTTP network port.
psc.as.HTTP.compress	on	Specifies whether HTTP compression support is enabled.

HTTPS connection properties

Set these HTTPS connection message-handling properties to increase network throughput:

Property	Default	Description
psc.as.HTTPs.maxconnections	-1	Maximum client connections on the HTTPS network port.
psc.as.HTTPs.compress	on	Specifies whether HTTPS compression support is enabled.

HTTP message properties

Set these HTTP message connection properties to increase network throughput:

Property	Default	Description
psc.as.msg.timeout	10000	Timeout for async requests in milliseconds.
psc.as.msg.maxpostsize	2097152	The maximum size in bytes of a POST HTTP message.
psc.as.msg.socketbuffer	9000	The HTTP message buffer size in bytes.
psc.as.compress.min	2048	The minimum message size in bytes that enables compression for HTTP responses.
psc.as.compress.types	text/html text/xml text/javascript text/css application/json	A comma-separated list of MIME types that can be HTTP compressed.

HTTP/HTTPS request processing

PAS for OpenEdge handles all HTTP client requests, including APSV, REST, SOAP, and WEB. Tuning concurrent client request handling involves managing the PAS for OpenEdge thread pool and the HTTP request queues. The goal is to tune the thread pool size to handle the maximum number of concurrently executing client requests across all web applications within the bounds established by the JVM configuration.

Property	Default	Description
psc.as.executor.maxthreads	300	Maximum number of threads that can be created to execute client requests.

Property	Default	Description
psc.as.executor.minsparethreads	10	Minimum number of threads retained in the pool to service client requests.
psc.as.HTTP.maxqueuesize	100	Maximum queue size for parking HTTP connection requests until a thread from the thread pool is available.
psc.as.HTTPs.maxqueuesize	100	Maximum queue size for parking HTTPS connection requests until a thread from the thread pool is available.

Tuning tips

- Each thread in the pool allocates memory from the JVM heap space. Having too many threads reduces the amount of JVM memory available to the applications. Having too few threads reduces the ability of the instance to handle concurrent client requests.
- Another factor to consider when setting the thread pool properties is the instance startup time. The larger the number of threads initially started (`minsparethreads`), the longer it takes for the instance to start and be available for clients. Not starting enough threads or keeping too few threads active, results in irregular client response times because new threads are created and initialized before they can be used.
- Monitoring for the purpose of determining these values is generally obtained during peak and non-peak client activity times. Look for client errors due to overloaded queues or thread pool exhaustion as indicators that values need to be increased. Look for irregular client response times as indicators that the minimum retained thread count is not sufficient.

PAS for OpenEdge instance parameters

PAS for OpenEdge comes preconfigured with certain server features and HTTP request filters (referred to as *valves*) that are executed on each client request. Although the default settings work for most situations, some server features or filters can be disabled to adjust security or HTTP request processing time. The recommended method of controlling server options is to use the PASMAN command-line utility:

```
pasman.{bat|sh} feature feature-name={on|off} -I instance-name
```

The complete list of instance features can be displayed by typing the following command:

```
proenv>pasman.{bat|sh} feature -I instance-name
VersionLoggerListener=off
SecurityListener=off
APRListener=off
PSClifecycle=on
JMXLifecycle=off
PSCRegistry=on
HealthCheck=off
HTTP=on
HTTPS=on
AJP13=off
Cluster=off
UserDatabase=on
JAASRealm=off
LDAPRealm=off
PASInstrument=off
RemoteHostValve=on
RemoteAddrValve=on
SingleSignOn=on
AccessLog=on
CrawlerSessionManager=on
StuckSessionValve=on
```

A description of each individual feature can be displayed by typing the following command:

```
pasman.{bat|sh} help feature-name -I instance-name
```

Tuning tips

- The `StuckSessionValve` request filter applies to all deployed web applications, including the OpenEdge web applications that execute the ABL language requests. This setting can result in false positives, so interpret the information as informational and not an error.
- The `CrawlerSessionManager` feature protects against large numbers of HTTP sessions being started by external web page indexers that consume a lot of heap space. For web-based PAS for OpenEdge instances, Progress recommends that you leave the feature enabled. If you have intranet PAS for OpenEdge instances, you can disable the feature.
- The `AccessLog` feature uses processing time to format and write the tracking of HTTP clients. This feature should be disabled unless deemed necessary.
- The `SingleSignOn` (SSO) feature allows the user to log in once to the PAS for OpenEdge instance for all web applications in a particular realm. Both the Apache Tomcat management and OpenEdge remote administration web applications share the same realm and benefit from SSO. The OpenEdge web applications (`oeabl.war`) can also benefit from this functionality if they are configured for container security. If your production PAS for OpenEdge instance does not deploy those administration web applications or need to benefit from SSO, then disable this feature.
- PAS for OpenEdge instance clusters are advanced web server architectures and the `Cluster` feature should remain disabled until it is necessary to support your ABL application using multiple PAS for OpenEdge instances.
- The `SecurityListener` feature is used infrequently but is available for best-practice production security configurations. Enable this feature only if you are certain that PAS for OpenEdge instances are deployed using the required OS file and directory permissions.

- The `RemoteHostValve` and `RemoteAddrValve` default configuration allows all host names and addresses, so their overhead is small. Both DNS and IP addresses are considered unreliable for identifying internet clients, so you can disable these if PAS for OpenEdge is web based. They may be of more use when PAS for OpenEdge is running on what the end user site considers a secured intranet.

Tune OpenEdge web applications and multi-session agents

After you tune PAS for OpenEdge client requests, you can begin tuning the OpenEdge web applications and multi-session agent processes to achieve the following goals:

- Control the size of the ABL session pool in the session manager
- Manage resource consumption within an multi-session agent process
- Manage concurrent ABL request execution
- Manage PAS for OpenEdge startup time

The design and implementation of your ABL application affects the PAS for OpenEdge instance and all web applications deployed on it. In most cases, the ABL application design and implementation should not be changed. Your goal is to balance your PAS for OpenEdge instance startup times, supporting client requests to your ABL business logic, and the response time to those clients.

Your OpenEdge web applications and the multi-session agent process execution of client requests are gated by the way you tune the instance in which they operate. For example, it does not matter if you configure OpenEdge to handle 400 concurrent client requests if the instance it runs in can support only 200 concurrent client requests.

Another reality is that the execution speed of the ABL language engine and the OpenEdge RDBMS storage engine are tied to the speed of the OS, CPUs, and file system, but not to the PAS for OpenEdge instance.

ABL session manager and session pool

The ABL session pool is managed by the session manager which is common to all of its configured OpenEdge web applications in a PAS for OpenEdge instance.

The properties specified in the *instance-name*`/conf/openedge.properties` configuration file control how the session manager manages the ABL session pool, the pool of multi-session agents, and the local network socket connections.

The file can be edited managed using the OEPROP command-line utility, OpenEdge Management, or a text editor.

This topic describes the OpenEdge properties and how they can be used to tune the multi-session agent and ABL sessions of a PAS for OpenEdge instance:

Property [AppServer.SessMgr]	Default	Description (per ABL application)
`numInitialAgents`	`1`	Number of multi-session agent processes to create at server startup.
`maxAgents`	`2`	Maximum number of multi-session agent processes that can exist.

Property [AppServer.SessMgr]	Default	Description (per ABL application)
maxABLSessionsPerAgent	200	Maximum number of ABL sessions per multi-session agent process.
maxConnectionsPerAgent	200	Maximum number of network connections between session manager and an multi-session agent.
requestWaitTimeout	15000	Maximum time in milliseconds that a client request waits for an ABL session before an error is generated.
idleSessionTimeout	300000	Maximum time in milliseconds that an ABL session remains idle before it is shut down.
idleAgentTimeout	1800000	Maximum time in milliseconds that a multi-session agent remains idle before it is shut down.
agentListenerTimeout	300000	Maximum time in milliseconds that the session manager waits for an multi-session agent to report it started before an error is generated.
agentStartLimit	0	Determines the maximum number of agents that can be started at the same time. If this value is zero or not defined, then no limit is enforced. This property can be used to prevent an "agents explosion", which is sometimes seen when a burst of new client activity occurs when no agent sessions are available on existing agents. In response to this, the session manager may try to start a new agent process for every client request received. The agentStartLimit property can cap the number of agents started simultaneously.
minAgents	1	Minimum number of agents expected at any time. If the number of agents drops below this number, then a client request causes the session manager to start enough agents to match this number. • Setting of 0 means minAgents is disabled. • Any value greater than 0 is used for minAgents. • If minAgents is greater than maxAgents, then the minAgent value is lowered to match the maxAgents. The session manager ensures that there is always one multi-session agent available at all times before running the request.

Property [AppServer.SessMgr]	Default	Description (per ABL application)
idleConnectionTimeout	300000	Timeout value in milliseconds between a PAS for OpenEdge client and the session manager. If a connection is idle for more than the specified timeout value, the session manager terminates the connection by automatically disconnecting the connection to the PAS for OpenEdge instance.
idleResourceTimeout	0	Timeout value in milliseconds to determine the frequency with which the PAS for OpenEdge instance checks for idle resources. Any resource (for example, a connection, agent, or client session) that was not accessed more recently than the specified timeout for that property is terminated. If this property is set to 0, then idle resource checking is disabled.
defaultAgentWaitAfterStop	30000	Timeout value in milliseconds to wait for a busy agent to finish work during a shutdown caused by the expiration of the idleAgentTimeout. This value should be greater than the normal execution time of the application's longest running request.
defaultAgentWaitToFinish	30000	Timeout value in milliseconds to wait for a stopping agent to finish work during a shutdown caused by the expiration of the idleAgentTimeout. When this timeout expires, the agent is forcibly shut down. This value should be greater than the longest time required to handle a normal STOP condition for the application.

Property [Appserver.Agent]	Default	Description (per multi-session agent)
numInitialSessions	2	Number of ABL sessions started at multi-session agent startup time.

Note: Full property descriptions are located in the *instance-name*/conf/openedge.properties.README file.

How the OpenEdge properties should be set depends on the operating model implemented by your ABL application:

- Stateful
- Stateless

These application model implementations affect property settings and require an understanding of how the ABL session manager manages the client session pool, the multi-session agents in which the ABL sessions run, and how ABL sessions are selected to execute client requests.

The first priority of the session manager is to conserve system resources by maximizing the pool of ABL sessions and by scaling a larger number than is needed to handle the current client load. PAS for OpenEdge uses a find-first-free scheduler and only one ABL session. It thereby consumes only one session's worth of OS memory and file system handles.

The following is a general description of how the session manager implements its find-first-free scheduling of client requests using multi-session agents, local socket connections, and ABL sessions:

1. The instance receives a client request from an `oeabl` web application transport.
2. The first multi-session agent in the list of multi-session agents is set as the current agent.
3. If the current multi-session agent has an unused agent connection and a free ABL session, the multi-session agent executes the client request.
   - During this time, the local agent connection is placed into an *in-use* state and the free ABL session is *bound* to the agent connection, so neither is available for use by other client requests.
   - When the client request ends, the ABL session is marked as *free* if a persistent procedure has not been executed and the ABL application code has not set a bound-client condition. The local agent connection returns to the *unused* state as available to execute any other client's request.
4. If the number of current multi-session agent local agent connections is less than the maximum, and any ABL sessions are free, the session manager creates a new agent connection to run requests on and executes the client request.
5. If the number of current multi-session agent local agent connections reaches the maximum, and another multi-session agent exists in the multi-session agent list, then the next multi-session agent is set as the current agent, and executes the client request.
6. If the maximum number of multi-session agents has been reached, create a new multi-session agent. It is added to the end of the multi-session agents list, set as the current agent, and executes the client request .
7. If you receive a large number of client requests, then queue the request if the maximum request queue size is not exceeded. Queued requests are executed as soon as a multi-session agent socket connection becomes available.

PAS for OpenEdge provides support for stateless and stateful application architecture models, where all client requests are handled stateless and the ABL application can temporarily bind the client and operate in a stateful manner. The implications of this determine how many ABL sessions and multi-session agents to configure.

Web application startup parameters

The following topic explains how to control PAS for OpenEdge startup times, ABL sessions for executing client requests, and concurrent client request execution.

Startup time and initial client request handling

When a PAS for OpenEdge instance is started, the session manager must start at least one multi-session agent. During the startup of a multi-session agent, the session manager holds until the multi-session agent gets a callback connection from the session manager to report its startup status and any subsequent notifications. The session manager holds for a `agentListenerTimeout` period before it considers the multi-session agent either unresponsive or unable to start.

The session manager uses the `numInitialAgents` property to begin building a list of multi-session agent processes. The multi-session agent process uses the `numInitialSessions` property at startup time to begin building the list of physical ABL sessions. For example: 1 multi-session agent with 5 initial ABL sessions yields an initial ABL session pool size of 5. Starting 3 multi-session agents with 5 initial ABL sessions each, yields an initial ABL session pool size of 15.

Tuning tips:

- The longer the time the `agentListenerTimeout` property is set, the longer the PAS for OpenEdge instance takes to start if the multi-session agent process has startup failures. If the `agentListenerTimeout` property is set to too short of a time, then the PAS for OpenEdge instance may start sooner, but the session manager may consider the multi-session agent unavailable.
- Setting the number of multi-session agent processes and physical ABL sessions to start at initialization time depends on how long the ABL application implementation takes to run its startup procedure. A larger number causes the PAS for OpenEdge instance to not handle incoming client requests for a longer period of time. A smaller number causes client requests handled by PAS for OpenEdge to have slow response times when new ABL sessions, or multi-session agent processes, or both are started.

Multi-session agent parameters and ABL sessions

After the initial startup the session manager automatically scales the number of ABL sessions according to the client load. Session manager scaling begins with adding ABL sessions to the first multi-session agent process until its `maxABLSessionsPerAgent` limit is reached. If additional ABL sessions are required to meet client demand, then the number of physical ABL sessions is scaled up in the next multi-session agent process until its `maxABLSessionsPerAgent` limit is reached. This scaling process continues until the `maxAgents` limit is met, when the session manager begins to return `no session available` errors to its clients.

Tuning tips:

- The setting of the multi-session agent maximum ABL sessions property is related to the OS process memory or file limits. Each ABL session that runs your ABL application code consumes memory and other process resources. Therefore, do not set the maximum ABL sessions to a number greater than what the OS process can support.
- The best performance is obtained by keeping the number of multi-session agents as small as possible by maximizing the number of ABL sessions and concurrent client requests for each multi-session agent.
- The more multi-session agents that are started, the higher the overhead in dispatching incoming stateless client requests for execution. The ideal case for stateless ABL application models is one multi-session agent and a few ABL sessions.
- Scaling to handle more multi-session agent processes is normal when the ABL application supports stateful client requests. The overhead for stateful clients is low because they are directly dispatched for execution to an exact ABL session within an exact multi-session agent, without the need to find an idle ABL session.

As stated previously, the goal of the session manager is to conserve resources. It does this by periodically searching for multi-session agent and client sessions that are idle in excess of the `idleAgentTimeout` and `idleSessionTimeout` periods, respectively. When an idle multi-session agent or client session exceeds the idle time, it is removed from service and is shut down gracefully.

Tuning tips:

- If your client traffic has frequent peaks and non-peak hours, then this may lead to thrashing, causing client response times decrease because multi-session agents and ABL sessions are harvested too soon, and the client must wait for new ones to be started.
- If the OpenEdge metrics indicate that too many multi-session agent and ABL session startups and idle resource shutdowns occur, then increase the time they can remain idle before being harvested.
- You also have the option of manually controlling the harvesting of idle multi-session agents and ABL sessions. To disable the automatic idle resource harvesting, set the `idleResourceTimeout` property to 0, and use the JMX or remote OpenEdge administration plugin to manually harvest the idle resources.

Manage concurrent request execution

A multi-session agent can concurrently execute as many client requests as it has socket connections between the session manager and itself. The session manager automatically scales the number of local socket connections to multi-session agents up to the `maxConnectionsPerAgent` limit while looking for a free ABL session to execute a client request. A result of managing the number of local socket connections is the multi-session agent management of its pool of OS threads reserved for executing client requests. Each time a new local socket connection is created, a new OS thread is started, to execute client requests arriving through that socket. Each time a local socket connection is closed an OS thread from the pool is stopped. When a local socket connection is created and an OS thread exists to execute client requests, it can be used by any client request regardless of the bound-client state.

Tuning tips:

- Do not set the `maxConnectionsPerAgent` value to be greater than the value for `maxABLSessionsPerAgent`.
- Do not set the sum of local socket connections for all multi-session agents to be greater than the PAS for OpenEdge thread pool for executing HTTP requests, because they will never be used.
- Choosing an appropriate number of local socket connections is closely tied to the implementation of the ABL application. If the ABL application uses high levels of file, database, and socket I/O, then increase the number of socket connections and ABL sessions because many of them will have high idle times, then the CPU resources can be used for executing CPU-bound ABL language statements.

Note: The starting and stopping of local socket connections and the OS threads in its thread pool is not related to the startup and shutdown of physical ABL sessions.

6

Load balance PAS for OpenEdge instances

A full-featured load balancer is a required part of any highly available and fault tolerant environment. Load balancing in PAS for OpenEdge is accomplished through industry-standard HTTP options. Load balancers are not an OpenEdge product, so finding the correct load balancer with the needed features is crucial. This content includes an overview of configuration tasks for each option, and where applicable, describes the pros and cons of the option.

There are several load balancer features that can be used with PAS for OpenEdge:

- Forwarding client requests to multiple PAS for OpenEdge instances
  - Spreads the workload over several instances
  - Quickly recover from an instance crash by starting a new VM or machine to automatically register with the load balancer
  - Upgrade the application without interruption with the following workflow:
    - Create a new PAS for OpenEdge instance with new code.
    - Add the instance to the load balancer.
    - Remove the old PAS for OpenEdge instance with old code.

- Load balancer health check
  - Verify the health of a PAS for OpenEdge instance.
  - Remove any instance that fails the health check.
  - Remove an instance before it fails with the OpenEdge HealthScanner.

- Auto Scaling load balanced PAS for OpenEdge instances
  - Scale up and down instances based on load (or other measure), for example:
    - Run the smallest size instance at non-peak times.
    - Scale up multiple instances to cover peak times.
    - Scale down again after peak times.

There are many third-party load balancing products available. The load balancing options covered in this content were tested with the latest OpenEdge release.

For details, see the following topics:

- Scaling considerations in PAS for OpenEdge
- Enable HTTP sessions for the APSV transport
- Clusters and sticky sessions
- Apache proxy host load balancing
- Tomcat load balancing
- Amazon load balancing
- Docker container load balancing

Scaling considerations in PAS for OpenEdge

PAS for OpenEdge is scalable; it can be configured to optimize performance as the workload increases. Note the steps for how you scale PAS for OpenEdge in the following areas:

- **Add more maxConnectionsPerAgent**—The first step in scaling an instance is to adjust the `maxConnectionsPerAgent` in order to maximize the response time of the instance under a given workload. You can set the `maxConnectionsPerAgent` using the OEPROP command-line utility.The methodology for tuning an instance is described in Tune PAS for OpenEdge instances.
- **Add and remove agents**—The next step in scaling an instance is to adjust the number of agents if you cannot add more `maxConnectionsPerAgent` but still have CPU and memory available. You can adjust the number of agents running on an instance using the OEPROP command-line utility. The methodology for adding agents is described in Tune PAS for OpenEdge instances.
- **Load balancing**—The last step in scaling is load balancing. Load balancing involves adding more instances of the server and configuring an environment where the workload is shared among those instances. Load balancing is achieved in PAS for OpenEdge through an Apache extension module or through third party applications, for example:
  - Apache proxy host load balancing
  - Tomcat load balancing
  - Amazon load balancing
  - Docker container load balancing

For information about machine sizing and to download a sample application that can be used for testing the tuning properties and size of the PAS for OpenEdge instance, see the PAS for OpenEdge ATM Benchmark Test and Machine Sizing Guide on the Progress Community.

Enable HTTP sessions for the APSV transport

To use HTTP sessions, ensure that the *`instance-name.application.transport`* `adapterEnabled` property is set to `1`.

HTTP sessions are required for load balancing. 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.

Note: You can use the `tcman sessions` action to display how many sessions are active for a web application. For more information, see Display web application HTTP sessions (sessions).

Applications deployed to REST, SOAP, and WEB transports implicitly use HTTP sessions. By default, the ROOT application for APSV is enabled. To enable HTTP sessions, set the following combination of *`instance-name.application`*`.APSV` properties:

1. Edit the *`instance-name`*`/conf/openedge.properties`.
2. Locate the `[`*`instance-name.application`*`.APSV]` entry for the application.

3. Enable `useHTTPSessions` and `enableRequestChunking` by setting them to `1`.

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

   Progress recommends that chunking is enabled when you enable the HTTP session to improve performance. 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 ensures that all the data gets transmitted in a single post operation in the context of a single session to a single server.

4. Save the changes.
5. Restart the instance for the server to apply the changes.

For an overview of all *`instance-name.application`*`.APSV` properties, see the APSV transport property descriptions in the `openedge.properties.README` file. If you want to implement load balancing, see Load balance PAS for OpenEdge instances on page 207.

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. 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 Use the SERVER-CONNECTION-CONTEXT attribute in *Application Development with PAS for OpenEdge* and SERVER-CONNECTION-CONTEXT attribute in *ABL Reference*.

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 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 `stickysession` 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:

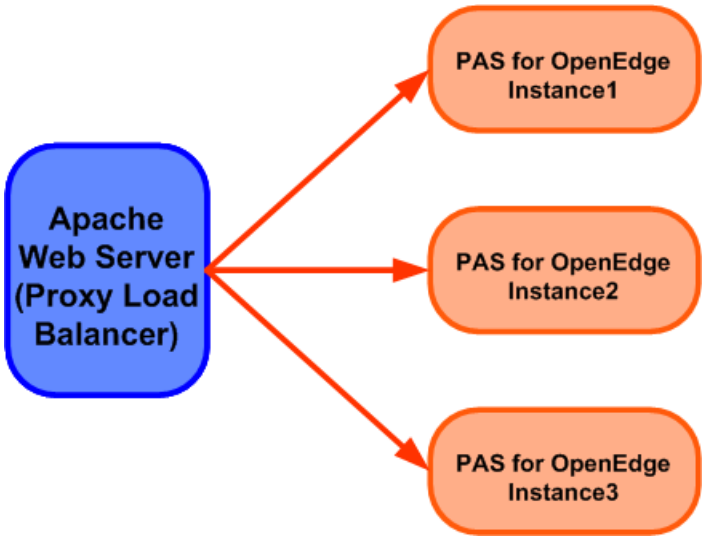
```
#Load balancer Proxy settings (session free)
      <Proxy "balancer://mylbgroup">
      BalancerMember "http://172.16.21.39:11600"
      BalancerMember "http://172.16.21.39:11610"
      BalancerMember "http://172.16.21.39:11620"
      </Proxy>
      ProxyPass "/test" "balancer://mylbgroup"
      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 effect on performance in a load balancing configuration because client requests can be bound to a single node among all available nodes.

Apache proxy host load balancing

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



Apache works as a proxy that re-directs 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.

For more information, see http://httpd.apache.org/docs/current/mod/mod_proxy.html.

Note: The disadvantage in this type of load balancing is that the Apache server does not monitor the running status of instances. If an instance crashes or is otherwise unavailable, then the Apache load balancer continues to try to send requests to the unavailable instance and requests are lost. Compare with Tomcat load balancing on page 215, where the running status of instances is monitored.

Requirements for Apache proxy load balancing

The requirements for implementing Apache proxy load balancing are:

- **Properly configured Apache HTTP Server**

  You configure a load balancing Apache HTTP Server (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; *sticky sessions* in Apache documentation), or to any available instance (session-free).

  If you run one or more web application available to the public, then install Apache in a demilitarized zone (DMZ) with the PAS for OpenEdge instances running behind a second firewall.

- **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.

Configure Apache 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 Apache, you must configure each PAS for OpenEdge instance. See Configure PAS for OpenEdge instances for proxy load balancing on page 214.

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 uncommenting `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 Apache 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**—After a session is started, all requests return to the same PAS for OpenEdge instance until the session is terminated.

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

If there is no need for client requests to return the the same PAS for OpenEdge instance (that is, 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, 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, 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, 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.

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

Configure 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 about how to enable chunking and HTTP sessions for the APSV transport.

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

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=
```

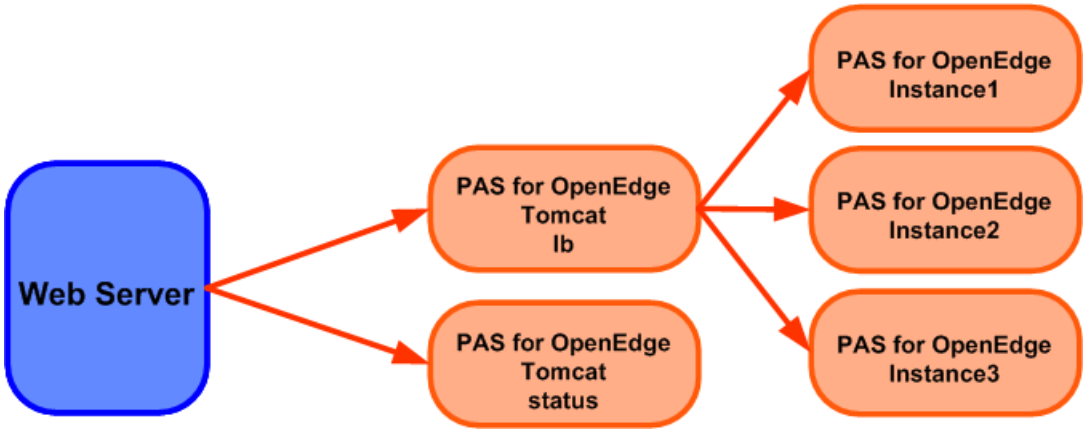
Note: The `jvmRoute` alias translates to the route value in the `http.conf` file of the Apache server. For more information, see Configure Apache for proxy load balancing on page 213.

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>  -->
```

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 show this load balancing configuration.



Some advantages of this configuration, compared to Apache proxy host load balancing on page 212, are:

- The PAS for OpenEdge instance that is designated as a load balancer (`lb` in the figure) is aware of the run-time 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 run-time statistics for each instance in the load balancing group (the `status` instance in the figure).

Requirements for Tomcat load balancing

The following are the requirements to implement Tomcat load balancing:

- **A properly configured web server.**

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

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

  Do not deploy any web applications to the load balancer.

- **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, and an optional status worker that monitors the run-time status of the other workers.

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

Configure an Apache web server for Tomcat load balancing

The 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 the Apache configuration is described because Apache is a common web server, and it is not practical to describe the configuration details of all the available web servers.

---

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

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

4. Update AJP connector section in `server.xml` with `secretRequired="false"`.

```
<!-- feature:begin:AJP13:on -->
    <Connector executor="tomcatThreadPool"
               address="${psc.as.ajp13.address}"
               port="${psc.as.ajp13.port}"
               protocol="AJP/1.3"
               asyncTimeout="${psc.as.msg.timeout}"
               maxPostSize="${psc.as.msg.maxpostsize}"
               acceptCount="${psc.as.http.maxqueuesize}"
               maxConnections="${psc.as.http.maxconnections}"
               tomcatAuthentication="${psc.as.ajp13.tomcatauth}"
               redirectPort="${psc.as.https.port}"
               URIEncoding="${psc.as.http.uriencoding}"
               secretRequired="false" />
    <!-- feature:end:AJP13:on -->
```

5. Update ajp13 address to receive connections from anywhere (Update if pasoe and load balancer are on different machines) - Update property in *instance*`/conf/catalina.properties` as follows:

```
psc.as.ajp13.address=::
```

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

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

Configure PAS for OpenEdge for Tomcat load balancing

To configure PAS for OpenEdge to enable Tomcat load balancing:

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

You can use the PASMAN `create` action to instantiate an instance. The instance 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. **(Optional) Create an instance (`status`) to gather metrics about the instances in the load balancing group.**

   You can use the PASMAN `create` action to instantiate an instance. The instance 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 run-time status of instances in its load balancing group. It will not forward a request to an instance that crashed or is not running. Therefore, the status instance is only required if you want to monitor performance data about the instances in the load balancing group.

3. **Generate and deploy a `worker.properties` file.**

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

   You can create a preliminary `worker.properties` file using the PASMAN `workers` action. The action gathers information about 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, then 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.

Amazon load balancing

Amazon Web Services offers an Elastic Load Balancing (https://aws.amazon.com/elasticloadbalancing/) product that distributes incoming application across multiple Amazon Elastic Compute Cloud (EC2) instances. 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 define. You can use these products together to create a cloud-based load balancing configuration for PAS for OpenEdge.

Configure 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 12: Elastic Load Balancing and Auto Scaling components

Component	Description
Virtual private cloud (VPC)	A private network of your application machines, which is your internal network that is not connected to 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, automatically starts 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:
   - Define a network mask range for your private network
   - Define 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 because they are running on different hosts and therefore have unique IP addresses in the VPC.
3. Create an ELB, and enable:
   - 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, which are required for ABL connections using the APSV transport to load-balanced PAS for OpenEdge instances.

     Note: If you want to ensure that you never lose the stickiness of an ABL connection to PAS for OpenEdge, set the sticky session timeout to `0`.
   - Cross-zone load balancing
   - Log timing
   - Idle timeout
   - 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 copies of the AMI that can be launched
   - A health check configuration
   - The scaling policy (which 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. For more complete information, see the Amazon documentation.

Docker container load balancing

PAS for OpenEdge supports running instances in Docker containers. Docker containers support clustering tools such as Docker Swarm and Kubernetes, which allow you to run multiple PAS for OpenEdge containers in a cluster. Clustering provides high availability and elasticity.

For more information about clustering and load balancing orchestration using containers, see the Kubernetes Documentation and the Docker Swarm documentation.

For more information about running PAS for OpenEdge in a Docker container, see "Run PAS for OpenEdge in a Docker container" in *Run PAS for OpenEdge in a Docker Container*.

7

Monitor PAS for OpenEdge instances

PAS for OpenEdge provides system administrators with numerous tools and ways to monitor their instances and applications in production. These monitoring tools include:

- The Logback logging framework which is supported by an enhanced Mapped Diagnostic Context (MDC) logging framework. For more information, see Configure logging in PAS for OpenEdge on page 224.
- JMX access using the Use OEJMX to manage and monitor an instance on page 244.
- REST APIs that are documented by the Use Swagger UI to explore management REST APIs on page 258 and in REST API Reference for oemanager.war.
- About the OpenEdge HealthScanner on page 259
- Use Server-Side ABL Performance Profiling on page 251
- Use deferred logging in PAS for OpenEdge on page 242
- Collect runtime metrics on page 279
- Enable debugging on page 289

For details, see the following topics:

- Configure logging in PAS for OpenEdge
- Use deferred logging in PAS for OpenEdge
- Use OEJMX to manage and monitor an instance
- Use Server-Side ABL Performance Profiling
- Use Swagger UI to explore management REST APIs

- About the OpenEdge HealthScanner
- Collect runtime metrics
- Enable debugging
- About session and request states
- Troubleshoot problems with an instance
- Find memory leaks using ABL object tracking

Configure logging in PAS for OpenEdge

OpenEdge implements the Logback framework and supports enhanced Mapped Diagnostic Context (MDC) logging. MDC allows the PAS for OpenEdge instances to dynamically inject request context information into the logging subsystems, where the request contexts can be exposed through tokens in log entry formats. The Logback framework allows for consistency across all of OpenEdge and Tomcat.

Note: The Logback framework is implemented for all log files except for the ABL application agent log file (*app-name*`.agent.`*{date}*`.log`), which uses the OpenEdge Logger framework. This log file is described in Configure ABL application agent logging.

In OpenEdge, the Logback XML files are localized into properties files that specify the following:

- Log file directory location
- Log file name pattern
- Log entry format patterns
- Default logging levels

Each property file contains the configurations for multiple OpenEdge or Tomcat components.

With MDC logging, the PAS for OpenEdge default log entry formats can include essential request context information in all log files, and allow for user-made customizations when additional context is needed.

This topic describes how to configure logging for all log files available to PAS for OpenEdge and Tomcat.

See also:

- Configure Tomcat and Tomcat web application logging
- Configure PAS for OpenEdge application logging
- Configure ABL application agent logging
- Configure log file patterns
- Configure MDC access for header requests

To learn about Elastic Fluentd Kibana (EFK) stack logging for PAS for OpenEdge in a Docker container, see Unified Logging for PAS for OpenEdge Container.

Configure Tomcat and Tomcat web application logging

The *instance-name*/conf/logging-tomcat.properties file sets the logging properties for Tomcat components.

The following Tomcat logging components can be configured in logging-tomcat.properties:

- catalina.out—Core Tomcat console logging output file.
- catalina.*date*.log—Tomcat server or any of the deployed web application activity log.
- localhost.*date*.log—Web application activity. The default Tomcat hostname log.
- manager.*date*.log—Activity of the Tomcat manager web application.
- host-manager.*date*.log—Activity of the Tomcat manager web application for virtual hosts.
- oehealth-collector.*date*.log—Activity of the OpenEdge HealthScanner web application.
- localhost-access.*date*.log—Activity of requests processed by the web server.

The following table defines the *instance-name*/conf/logging-tomcat.properties configuration file that controls the logging for the Tomcat server and its distributed web applications:

Tomcat subsystem	Log filename	Property name	Value definition
Java standard output (stdout)	catalina.out	psc.as.logging.console.pattern	Defines the output format for what is logged, in which columns, and with what static content.
		psc.as.logging.console.level	Defines the logback logging level for all console logging.
Tomcat server	catalina.*date*.log	psc.as.logging.catalina.fileName	Defines the output filename format and how often new instances of the log file are created.
		psc.as.logging.catalina.level	Defines the amount of logging detail to output for the subsystem.
Tomcat server client connectors	localhost-access.*date*.log	psc.as.logging.access.fileName	Defines the output file name format and how often new instances of the log file are created.
		psc.as.logging.access.pattern	Defines the output format for what is logged, in which columns, and with what static content.

Tomcat subsystem	Log filename	Property name	Value definition
Tomcat server virtual host	`localhost.`*`date`*`.log`	psc.as.logging.localhost.fileName	Defines the output filename format and how often new instances of the log file are created.
		psc.as.logging.localhost.level	Defines the amount of logging detail to output for the subsystem.
Tomcat manager web app	`manager.`*`date`*`.log`	psc.as.logging.manager.fileName	Defines the output filename format and how often new instances of the log file are created.
		psc.as.logging.hostmanager.level	Defines the amount of logging detail to output for the subsystem.
Tomcat OpenEdge HealthScanner	oehealth-collector.*date*.log	psc.as.logging.oehealth.fileName	Defines the output filename format and how often new instances of the log file are created.
		psc.as.logging.oehealth.pattern	Defines the output format for what is logged, in which columns, and with what static content.
		psc.as.logging.oehealth.level	Defines the amount of logging detail to output for the subsystem.
Tomcat server common	Common to all Tomcat log files	psc.as.logging.tomcat.pattern	A common log file output format if one is not explicitly defined in the logging subsystem.
		psc.as.logging.maxhistory	The maximum number of log file versions that will be kept in the logging destination directory.
		psc.as.logging.tomcat.level	Common logging level if not explicitly defined in the above logging subsystems.

Note:

- Logback product logging levels include: OFF | ERROR | WARN | INFO | DEBUG | TRACE
- Logback product logging pattern token definitions are fully described in the Logback product documentation.
- The property value can include tokens for MDC, OS process environment variables (case sensitive), and properties defined in the logging property files

Configure PAS for OpenEdge application logging

The *instance-name*/conf/logging-pasoe.properties file sets the logging properties for PAS for OpenEdge application components.

The logging configuration for ABL applications is split between the PAS for OpenEdge Tomcat server and the ABL agent processes which execute the business application. Each deployed ABL application has one single set of log files.

PAS for OpenEdge also enables application developers to write custom logs that can be configured by an administrator. To learn more about custom ABL application logging, see ABL application logging.

The following PAS for OpenEdge logging components can be configured in logging-pasoe.properties:

Log filename	Required	Configuration file	Notes
abl-app-name.*date*.log	Yes	*instance-name*/conf/logging-pasoe.properties	Fully configurable output format and MDC content.
abl-app-name.agent.*date*.log	Yes	• *instance-name*/conf/logging-pasoe.properties • *instance-name*/bin/agent.[sh|bat]	Restricted ability to configure the output format. Described more fully in Configure ABL application agent logging.
*abl-app-name*_authn.*date*.log	No	*instance-name*/conf/logging-pasoe.properties	Created when the Spring Security user authentication event logging is enabled for the ABL application.
*abl-app-name*_authz.*date*.log	No	*instance-name*/conf/logging-pasoe.properties	Created when the Spring Security user URL authorization event logging is enabled for the ABL application.

ABL application Spring Security and Session Manager configuration

The *instance-name*/conf/logging-pasoe.properties file contains the following logging configuration properties for the ABL application Spring Security and Session Manager configuration:

Log filename	Property name	Value definition
abl-app-name.*date*.log	psc.as.logging.oeabl.fileName	Defines file name format how often a new log file instance is generated.
	psc.as.logging.oeabl.pattern	Defines the output format that contains MDC, static text, and Logback product-defined tokens.
	psc.as.logging.oeabl.default	Defines the default logging level specified in Logback logging levels.

The *instance-name*/conf/logging-pasoe.properties file contains the following logging configuration properties for the ABL web application login event (authentication) log:

Log filename	Property name	Value definition
*abl-app-name*_authn.*date*.log	psc.as.logging.authn.fileName	Defines the output filename format.
	psc.as.logging.authn.pattern	Define the output format that contains MDC, static text, and Logback product-defined tokens.
	psc.as.logging.authn.level	Disable logging (OFF), record only failures (ERROR), or record both success and failures (INFO).

The *instance-name*/conf/logging-pasoe.properties file contains the following logging configuration properties for the ABL web application URL access event (authorization) log:

Log filename	Property name	Value definition
*abl-app-name*_authz.*date*.log	psc.as.logging.authz.fileName	Defines the output filename format.
	psc.as.logging.authz.pattern	Defines the output format for MDC, static text, and Logback product-defined tokens.
	psc.as.logging.authz.level	Disable logging (OFF), record only failures (ERROR), record both success and failures (INFO).

Configure web application logging in PAS for OpenEdge

PAS for OpenEdge includes a number of optional Java web applications that can be manually deployed and used in development, troubleshooting, and monitoring processes. The following table shows the *instance-name*/conf/logging-pasoe.properties file:

Default logfile name	Web application	Property name	Value definition
oemanager.*date*.log	oemanager.war	psc.as.logging.oeman.fileName	The output filename format for the oemanager web application, and defines how often a new log file instance is created
		psc.as.logging.oeman.pattern	The output format that contains static text, and Logback product-defined tokens.
		psc.as.logging.oeman.level	The default logging level specified in Logback logging levels.
oests.*date*.log	oests.war	psc.as.logging.oests.fileName	The output filename format for the OpenEdge Authentication Gateway server Security Token Service web application, and defines how often a new log file instance is created.
		psc.as.logging.oests.pattern	The output format that contains static text, and Logback product-defined tokens.
		psc.as.logging.oests.level	The default logging level specified in Logback logging levels.
oehealth.*date*.log	oehealth.war	psc.as.logging.oehealth.fileName	The output filename format for the oehealth web application UI, and defines how often a new log file instance is created.
		psc.as.logging.oehealth.pattern	The output format that contains static text, and Logback product-defined tokens.
		psc.as.logging.oehealth.level	The default logging level specified in Logback logging levels.

Default logfile name	Web application	Property name	Value definition
oedbg.*date*.log	oedbg.war	psc.as.logging.oedbg.fileName	The output filename format for the oedbg web application, and defines how often a new log file instance is created.
		psc.as.logging.oedbg.pattern	The output format that contains static text, and Logback product-defined tokens.
		psc.as.logging.oedbg.level	The default logging level specified in Logback logging levels.
oeproxy.*date*.log	oeproxy.war	psc.as.logging.oeproxy.fileName	The output filename format for the oeproxy web application, and defines how often a new log file instance is created.
		psc.as.logging.oeproxy.pattern	The output format that contains static text, and Logback product-defined tokens.
		psc.as.logging.oeproxy.level	The default logging level specified in Logback logging levels.

Configure ABL application logging

The OpenEdge 12 series of releases updates the way the ABL application and ABL web application logging is configured from previous releases of OpenEdge. In OpenEdge 12, ABL application logging is configured through the `logging-ablapp.xml` file.

Prior to OpenEdge 12, package-level application logging was configured per each ABL web application that is deployed to *instance-name*`/webapps/` directory, through the *instance-name*`/webapps/`*webapp-name*`/WEB-INF/logging.xml` file. This did not produce a common log output to the ABL application log file (*instance-name*`/logs/`*ablapp_name*`.`*date*`.log`), and made troubleshooting difficult.

In the OpenEdge 12 releases, the package-level application logging configuration still initializes from each deployed OpenEdge web application's `WEB-INF/logging.xml` file. However, this `logging.xml` file acts only as a bootstrap loader, which loads other logging configuration and property files that are common for the entire ABL application. You do not edit the bootstrap loader `WEB-INF/logging.xml` file for an ABL web application.

To enable ABL application package-level logging for a specific ABL application hosted on the instance, modify the *instance-name*`/ablapps/`*ablapp-name*`/conf/logging-ablapp.xml` file. Uncomment all of the packages that you want to generate logging information about, and specify the logging level.

The package-level application logging information is printed to the *instance-name*/logs/*ablapp_name.date*.log file for each enabled ABL application.

The logging.xml file in OpenEdge 12

The *instance-name*/webapps/*webapp-name*/WEB-INF/logging.xml loads the *instance-name*/conf/logging-pasoe.properties file that contains shared logging properties for all deployed ABL applications. Logging properties consist of the output file format, the output filename format, and more. You can edit the*instance-name*/conf/logging-pasoe.properties to change the default log file naming format, the frequency of how often the log files are rolled, and log file entry format.

The *instance-name*/webapps/*webapp-name*/WEB-INF/logging.xml bootstrap then loads the *instance-name*/conf/logging-oeabl.xml configuration, which holds information common to all deployed OpenEdge ABL web applications (that is, web applications derived from oeabl.war located in the *instance-name*/webapps directory). The information inside of logging-oeabl.xml uses properties from *instance-name*/conf/logging-pasoe.properties to ensure that log files and their contents are consistent across the instance. You do not edit the conf/logging-oeabl.xml file.

The conf/logging-oeabl.xml configuration file then loads the ABL application-specific logging directives from the *instance-name*/ablapps/*ablapp-name*/conf/logging-ablapp.xml file. The logging-ablapp.xml file is where you choose which OpenEdge packages should produce logging output and at what level of granularity.

In OpenEdge 12, there is one logging configuration for the entire ABL application, regardless of how many OpenEdge ABL web applications are deployed to it. This one logging configuration ensures that you cannot misconfigure any one OpenEdge web application's WEB-INF/logging.xml file, and corrupt your ABL application's log files. You do edit the ABL application's *instance-name*/ablapps/*ablapp-name*/conf/logging-ablapp.xml file.

The WEB-INF/logging.xml then loads the *instance-name*/conf/logging-security.xml configuration for generating the ABL application's authentication and authorization logs. Again, this is a common configuration so that all ABL application's security log file names and contents are consistent across all deployed ABL applications. You do not edit the conf/logging-security.xml configuration file. If you want to adjust the logging filename or log entry formats, then use the*instance-name*/conf/logging-pasoe.properties configuration properties.

Configure ABL application agent logging

The PAS for OpenEdge ABL application agent log file (*abl_app*.agent.*{yyyy-mm-dd}*.log) uses the OpenEdge Logger framework, rather than the Java-based Logback framework utilized by the other Tomcat and PAS for OpenEdge application log files. The ABL application agent log file records the activity of a multi-session agent.

The ABL application agent log filename and path are specified in the agentLogFile property in the *instance-name*/conf/openedge.properties.

Configure ABL application agent log output

The ABL application agent log file has limited control over the log file output. You can select enhanced or classic logging using the ENHLOGGINGOPTS environment variable specified in *instance-name*/bin/openedge_setenv.[sh|bat]. The default is enhanced logging.

The enhanced logging format supports compressed output format, client request ID logging, and ISO date-time formats while the classic logging format does not.

The value of the ENHLOGGINGOPTS can be set to one of the following configuration options:

Value	Behavior
undefined	Removing or commenting out the `ENHLOGGINGOPTS=; export ENHLOGGINGOPTS` statement enables classic logging behavior.
blank	The default formatting behavior is `ED`.
`D`	Formats date-time information using the ISO date-time standard. This value is ignored if classic logging is configured.
`d`	Formats date-time information using the classic logging format.
`E`	Operates in enhanced logging mode.
`e`	Operates in classic logging mode.

Note:

- If the message only comes from the internal multi-session agent code, then the request ID output appears as `?:?:?` or `mtapsv:-:?`.
- If the message comes from the deferred log buffer, then the request ID output appears as `-:-:-` followed by the `DFRD` component message.
- When a message is output during execution of code from a transport request, then the `OERequestInfo` class is available, and the expected values are provided for web application, transport, and request ID.

ABL application agent log file output description

The ABL application agent log file output begins when a multi-session agent starts with the following entry:

```
2019-08-23T10:55:16.650-0400 011476 027348 1 AS-Aux-0 mtapsv:-:? MSAS Agent Starting
Up -- Progress OpenEdge Release 12.1.
```

An ABL application agent log entry consists of eight standard components (more when deferred logging is enabled), for example:

	Log entry timestamp	Process ID	Thread ID	Logging level	ABL session	Request ID	Module or component writing the message	Log message
Examples	[illegible]	011476	027348	1	AS-Aux-0	mtapsv:-:?	MSAS	Agent Starting Up -- Progress OpenEdge Release 12.1.
	[illegible]	011476	012512	1	AS-7	ROOT:w:00000001	LogMgrWrtr	[illegible] INFO] Service "dataService v1.0.0" registered
	[illegible]	011476	028508	2	DFRD	-:-:-	DFRD	Deferred log start

The request ID is in the format `webapp:transport:requestID`. The `requestID` value is a left-padded hex, for example, `ROOT:w:0000000a`.

The transport in the request ID is identified by the following abbreviations:

- `w`—WEB transport
- `r`—REST transport
- `a`—APSV transport
- `s`—SOAP transport
- `f`—Static file

The ABL application agent log file also contains entries from internal ABL sessions and threads that do not correspond to actual PAS for OpenEdge client requests in the log entry column where the request ID is found. The following special request ID values are used to denote logging entries that originate from the multi-session agent at run-time:

Request ID	Description
`mtapsv:-:?`	The ABL session assigned to the client request is initializating. • The request originator is `mtapsv`. • The transport is `-` (n/a). • The client message number is ? (unknown).
`?:?:?`	The agent received a client request for processing, but a client request was not passed to the agent. The request continues to execute without client request tracking. • The request originator is ? (unknown). • The transport is ? (unknown). • The client message number is ? (unknown).

ABL application agent deffered log file output description

Deferred logging is a special in-memory buffer that prints to the ABL application agent log file when an agent crashes or when it is invoked through the `flushDeferredLog` API. For more information, see Use deferred logging in PAS for OpenEdge.

When the deferred logging buffer is invoked, it prints the following message to the ABL application agent log file:

```
2019-08-23T11:12:48.195-0400 011476 028508 2 DFRD -:-:- DFRD Deferred log start
```

This entry is followed by the deferred logging buffer that can be configured to print debug-level 4GLTrace information. This logging level provides a closer look at the ABL application agent processes.

Configure log file patterns

The default output patterns for the Tomcat server and the PAS for OpenEdge application components are contained in *instance-name*`/conf/logging-tomcat.properties` and *instance-name*`/conf/logging-pasoe.properties`, respectively. You can edit the output patterns to capture any Logback information you need.

The output pattern describes:

- What information is logged
- The order in which the information is logged
- What static content is logged

Mapped Diagnostic Context (MDC) tokens are also supported in log output patterns. MDC allows the PAS for OpenEdge instances to dynamically inject request context information into the logging subsystems, where the request contexts can be exposed through tokens in log entry formats.

This topic describes the defaults and supported values for the output pattern of each log file.

Tomcat and Tomcat web application pattern tokens

Logging output patterns, and designated log file names, where applicable, can be specified for the following Tomcat server components in the *instance-name*/conf/logging-tomcat.properties file:

```
# Core Tomcat console logging
psc.as.logging.console.pattern=%date{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} %-5level
[%thread] %logger{20} - %msg %n

# OpenEdge oehealth scanner logging
psc.as.logging.oehealth.fileName=oehealth-collector.%d{yyyy-MM-dd}.log
psc.as.logging.oehealth.pattern=%date{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} %-5level
[%thread] %logger{36} - %msg%n

# Core Tomcat client HTTP access loggging
psc.as.logging.access.fileName=localhost-access.%d{yyyy-MM-dd}.log
psc.as.logging.access.pattern=%h %l %reqAttribute{OEReq.userId}
[%date{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"}] "%r" %s %b %reqAttribute{OEReq.requestId}
%header{Accept} %n

# Common Tomcat & installed product properties
psc.as.logging.tomcat.pattern=%date{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} %-5level
[%thread] %C{36}.%M - %msg%n
```

The logging pattern supports Logback pattern tokens as described in the Logback Pattern Layout documentation. If you make changes the pattern, end the pattern with %n to create a new line after the log entry is written.

The Tomcat access log (`localhost-access.`*`date`*`.log`) supports all standard Logback layout fields as described in the Logback documentation. The following MDC fields can provide useful diagnostic information about your HTTP requests:

Table 13: Tomcat access log supported MDC fields

MDC field	Description
`%reqAttribute{OEReq.requestId}`	Tracks client HTTP requests through the Tomcat, ABL session manager, and multi-session agent.
`%reqAttribute{OEReq.userId}`	Replaces the Tomcat logging token %r with the OpenEdge Spring authentication user ID.
`%header{`*`http-header`*`}`	Any standard Tomcat HTTP request headers can be included into your access log.
`%responseHeader{`*`http-header`*`}`	Any standard Tomcat HTTP response headers can be included into your access log.
`%reqCookie{`*`cookie`*`}`	Requests the cookie name associated with the session.
`%D`	Provides the elaspsed time in miliseconds of the request.

Note: Providing invalid filter patterns can result in no localhost access log creation.

The standard output format of an MDC log across components consists of three colon-separated values in the following order: **webapp:transport:requestId**. The `requestId` value is left-padded hex, so an example output could be the following: **ROOT:w:0000000a**.

For the transport, the following abbreviations are used to determine the source:

- W = Web
- R = REST
- A = APSV
- F = File

For any request, the Tomcat Access log outputs this format exactly as stated if the pattern was modified.

PAS for OpenEdge application pattern tokens

Logging output patterns can be specified for the following PAS for OpenEdge application components in the *`instance-name`*`/conf/logging-pasoe.properties`:

```
# OpenEdge ABL webapp (oeabl.war) SessionManager & Spring Security logging
psc.as.logging.oeabl.fileName=${CONTEXT_NAME}.%d{yyyy-MM-dd}.log
psc.as.logging.oeabl.pattern=%date{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} %-5level [%thread]
%X{OEReq.requestId:--} %logger{36} - %msg %n

# OpenEdge ABL webapp login event logging
psc.as.logging.authn.fileName=${CONTEXT_NAME}_authn.%d{yyyy-MM-dd}.log
psc.as.logging.authn.pattern=%date{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} ${contextName}
%X{OEReq.requestId} %X{OESec.eventStatus} %X{OESec.loginPrincipal} - %X{OESec.failReason}
```

```
 -
%X{OESec.failDetails}%n

# OpenEdge ABL webapp URL access event logging
psc.as.logging.authz.fileName=${CONTEXT_NAME}_authz.%d{yyyy-MM-dd}.log
psc.as.logging.authz.pattern=%date{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} ${contextName}
%X{OEReq.requestId} %X{OESec.eventStatus} %X{OESec.authzUserId} [%X{OESec.failObj}]
%X{OESec.failAttrs} - %X{OESec.failReason}%n

# OpenEdge remote manager web application
psc.as.logging.oeman.fileName=${CONTEXT_NAME}.%d{yyyy-MM-dd}.log
psc.as.logging.oeman.pattern=%date{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} %-5level [%thread]
%logger{36} - %msg%n

# OpenEdge STS service ( for OpenEdge Authentication Gateway )
psc.as.logging.oests.fileName=${CONTEXT_NAME}.%d{yyyy-MM-dd}.log
psc.as.logging.oests.pattern=%date{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} %-5level [%thread]
%logger{36} - %msg%n

# OpenEdge PASOE health scanner service
psc.as.logging.oehealth.fileName=${CONTEXT_NAME}.%d{yyyy-MM-dd}.log
psc.as.logging.oehealth.pattern=%date{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} %-5level [%thread]

%logger{36} - %msg%n

# OpenEdge PASOE remote debugger service
psc.as.logging.oedbg.fileName=${CONTEXT_NAME}.%d{yyyy-MM-dd}.log
psc.as.logging.oedbg.pattern=%date{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} %-5level [%thread]
%logger{36} - %msg%n

# OpenEdge PASOE reverse-proxy service
psc.as.logging.oeproxy.fileName=${CONTEXT_NAME}.%d{yyyy-MM-dd}.log
psc.as.logging.oeproxy.pattern=%date{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} %-5level [%thread]

%logger{36} - %msg%n
```

The logging pattern supports Logback pattern tokens, as described in the Logback Pattern Layout documentation. If you make changes the pattern, then end the pattern with %n to create a new line after the log entry is written.

The OpenEdge ABL web application log file (*abl-app-name*.*date*.log), which includes the Session Manager and Spring Security logs, supports the following MDC fields which are designated by the %X token:

Table 14: Supported HTTP Request MDC Field Names for Spring Security and Session Manager logs

Logging field name	Logging field description	Tomcat access log syntax	
OEReq.userId	The Spring Security authenticated user ID, that is identified as the client from the server.	%{OEReq.userId}r	%X{
OEReq.sessionId	The HTTP session ID assigned to the current PAS for OpenEdge client login session (if server login sessions are configured)	%S	%X{

Logging field name	Logging field description	Tomcat access log syntax	
`OEReq.requestId`	The PAS for OpenEdge assigned request ID relative to the server instance (Note: This property is not unique across load-balanced server instances.)	`%{OEReq.requestId}r`	%X{
`OEReq.webapp`	The Java web application context name the PAS for OpenEdge client request executes in	n/a	%X{
`OEReq.transport`	The {apsv | soap | rest | web | file} PAS for OpenEdge service interface transport name the client's request executes in	n/a	%X{
`OEReq.remoteHost`	This field contains the remoteHost DNS address obtained from the PAS for OpenEdge network connection	`%h`	%X{
`OEReq.remoteAddr`	This field contains the remoteHost IP address obtained from the PAS for OpenEdge network connection	`%a`	%X{
`OEReq.serverName`	This field contains the name of the PAS for OpenEdge server	`%v`	%X{
`OEReq.URI`	This field contains the HTTP request's relative URI within `OEReq.webapp`	n/a	%X{
`OEReq.protocol`	This field contains the HTTP request scheme (http or https)	`%H`	%X{
`OEReq.method`	This field contains the HTTP request's method name	`%m`	%X{
`OEReqHead.`**`xxxxxx`**	This field contains the value of the HTTP request Header named **xxxxxx** or n/a if the header does not exist.	`%{xxxxxx}i`	%X{

Formatting Options for MDC values in ABL Spring and Session Manager logs

MDC values can be written to print a certain value if the requested information is not available. This is useful for troubleshooting. If an error occurs before the request ID, or other MDC field, is created, then the log can print a specified value.

The formatting options are written in this format:

```
%X{logging-field-name[:-default]}
```

For example:

%X{OEReq.requestId}	The log shows the assigned request ID or blank if the logging occurs before a request ID can be created.
%X{OEReq.requestId:--}	The log shows the assigned request ID or "-" if the logging occurs before a request ID can be created.
%X{OEReq.requestId:-na}	The log shows the assigned request ID or "na" if the logging occurs before a request ID can be created

Supported MDC field names for Spring Security authentication and authorization logs

The Spring Security authentication and authorization audit logs monitor for different events. Spring Security Authentication monitors for login pass/fail events. Spring Security authorization monitors for URL access events.

Spring Security Authentication logging fields

The OpenEdge ABL web application login event log file (*abl-app-name*_authn.*date*.log supports the following MDC fields that are designated by the %X token:

MDC field name	Logging field description	ABL application log synta
OESec.eventType	The type of Spring Security process being reported: "Authentication"	%X{OESec.evnetType}
OESec.eventStatus	The results of the client authentication process: { "success" | "failure" }	%X{OESec.eventStatus}
OESec.loginPrincipal	The client asserted account name for authentication	%X{OESec.loginPrincipal}
OESec.failReason	A simple string holding the reason for the authentication failure	%X{OESec.failReason}
OESec.failDetails	A more verbose string hold additional failure information	%X{OESec.failDetails}
OEReq.xxxxxx	The value of the HTTP request.	%X{OEReq.xxxxxx}
OEReqHead.xxxxxx	The value of the HTTP request header.	%X{OEReqHead.xxxxxx}

Note: For more information about OEReq and OEReqHead MDC field names, see the **Supported HTTP Request MDC Field Names for Spring Security and Session Manager logs** table.

Spring Authorization logging fields

The OpenEdge ABL web application URL access event log file (*abl-app-name*_authz.*date*.log supports the following MDC fields that are designated by the %X token:

MDC field name	Logging field description	ABL application log synta
OESec.eventType	The type of Spring Security process being reported: "Authorization"	%X{OESec.eventType}
OESec.eventStatus	The results of the client authentication process: { "granted" | "denied" }	%X{OESec.eventStatus}
OESec.authzUserId	The authenticated client's account name whose ROLEs are tested for URL access	%X{OESec.authzUserId}
OESec.failObj	The URL resource name being authorized	%X{OESec.failObj}
OESec.failAttrs	The authenticated client's account attributes used for authorizing URL access	%X{OESec.failAttrs}
OEReq.xxxxxx	The value of the HTTP request.	%X{OEReq.xxxxxx}
OEReqHead.xxxxxx	The value of the HTTP request header.	%X{OEReqHead.xxxxxx}

Note: For more information about OEReq and OEReqHead MDC field names, see the **Supported HTTP Request MDC Field Names for Spring Security and Session Manager logs** table above.

HTTP Request header fields are also supported for each audit log. For more information, see Configure MDC access to HTTP request headers.

Configure MDC access for header requests

Access to HTTP header values as Mapped Diagnostic Context (MDC) fields can be set using the OEReqHead.xxxxx token, where xxxxx is the HTTP header name (not case sensitive). The OpenEdge MDC support does not automatically make all HTTP headers available as MDC values. You must configure the list of header names if that level of information is required in your log files.

To configure the list of HTTP headers available as MDC fields:

1. Edit the PAS for OpenEdge instance *instance-name*/conf/oeablSecurity.properties file, so the list is available to all deployed ABL applications and their ABL web applications.
2. Append the MDC.filter.headerList property with a comma-separated list of header names:

```
MDC.filter.headerList=HeaderName1, HeaderName2
```

Note: Providing invalid header patterns can result in the no creation of the localhost access log.

3. Save the oeableSecurity.properties file.

   The HTTP request header name is available for use in %header fields for the localhost access log in *instance-name*/conf/logging-tomcat.properties, for example:

```
psc.as.logging.access.pattern=%h %l %reqAttribute{OEReq.userId}
[%date{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"}] "%r" %s %b %reqAttribute{OEReq.requestId}
%header{HeaderName1} %header{HeaderName2} %n
```

Configure the log file directory path

The default location for each of the PAS for OpenEdge log files is *instance-name*/logs.

You may need to change the log file directory path if you need a centralized place for your PAS for OpenEdge log files that is outside of the instance directory. This may be the case, for example, if you are running mutliple PAS for OpenEdge instances in Docker containers.

To configure the log file directory path, follow these steps:

1. Run the Proenv utility.
2. Run the PASMAN config command to view the catalina.logging.folder property, for example:

```
proenv>pasman.{sh|bat} config -I oepas1 catalina.logging.folder
catalina.logging.folder=${CATALINA_BASE}/logs
```

3. To change the log directory location, run the same command, but assign the new location to the catalina.logging.folder property. Then, repeat the command in Step 2 to confirm the output, for example:

```
proenv>pasman config -I oepas1 catalina.logging.folder=/oelogs/
proenv>pasman config -I oepas1 catalina.logging.folder
catalina.logging.folder=/oelogs/
```

Configure log rollover settings

PAS for OpenEdge Tomcat-level and application-level log files have the ability to rollover as specified in the `fileName` property in the `logging-tomcat.properties` and `logging-pasoe.properties` files in the `/conf` directory of the instance.

The log files defined in `logging-tomcat.properties` and `logging-pasoe.properties` adhere to the Logback framework. By default, the log `fileNames` definition includes `%d{yyyy-MM-dd}`, which indicates the log file rolls over at the start of each day. Log file rollover means the component of the instance stops writing to one log file and begins writing to a new log file.

The PAS for OpenEdge ABL application agent log file (*`abl_app`*`.agent.{yyyy-mm-dd}.log`) uses the OpenEdge Logger framework, rather than the Logback framework. The ABL agent log filename and path is specified in the `agentLogFile` property in *`instance-name`*`/conf/openedge.properties`.

The ability for the ABL application agent log to rollover depends on the inclusion of `{yyyy-mm-dd}` in the `agentLogFile` definition, for example:

```
agentLogFile=${catalina.base}/logs/oepas1.agent.{yyyy-mm-dd}.log
```

To stop ABL application agent log rollover, omit the date stamp in the log filename definition, for example:

```
agentLogFile=${catalina.base}/logs/oepas1.agent.log
```

Use deferred logging in PAS for OpenEdge

Deferred logging in PAS for OpenEdge opens an in-memory buffer to record information immediately preceding a multi-session agent crash. It also lets system administrators run an on-demand monitoring check on the instance. The in-memory buffer is configured independently from the agent log and automatically creates a `pasoe-agent-crash-trace-`*`process-id`* file in the event of a failure or when it is called through an API.

Additionally, a system administrator or DevOps engineer can use a REST or JMX API to dump the in-memory buffer to the agent log file for close monitoring of a production system.

The deferred logging is set in the `openedge.properties` file.There are three properties that allow system administrators to control how much in-memory logging information is stored: `defrdLogNumLines`, `defrdLoggingLevel`, `defrdLogEntryTypes`. The deferred logging buffer is enabled if all three properties are set to a non-zero value.

Deferred logging keeps a separate memory buffer from the regular agent logging. Deferred logging can be set to record different log entry types and to record at different logging levels. A separate memory buffer can help keep the regular agent log to a minimal size, while providing another type of logging for reconstructing agent crashes or monitoring the application without affecting performance.

For example, if you ran 200 sessions on each agent and ran multiple agents, it would require thousands of buffer lines to capture all the debugging information. This circumstance would create a log file that would be difficult to parse and could affect performance. Deferred logging allows you to create a separate logging buffer that saves only the specified number of log number lines and log entry types. This gives a higher logging level only immediately preceding an agent crash or when it is invoked through the API. Deferred logging allows for only the context of the crash to display, while allowing the regular agent log to print other messages or at a lower logging level.

To enable deferred logging on your PAS for OpenEdge instance:

1. Use the OEPROP utility to specify the number of lines to be saved in the deferred logging buffer:

```
oeprop.{bat|sh} AppServer.SessMgr.defrdLogNumLines=500
```

   There are two types of log entries that can be written to the buffer: Unconditional entries, including start-up and error events, and conditional entries, based on the logging level and startup type. The unconditional entries use some lines specified with this property. The default value is 0, which disables the feature.

2. Use the OEPROP utility to specify the logging level of the deferred logging buffer:

```
oeprop.{bat|sh} AppServer.SessMgr.defrdLoggingLevel=4
```

   You could use the deferred logging buffer to act as a stack trace preceding crash events, or set it to record a separate log entry type from the agent log. If this property is set to 0, then no deferred log file buffer is written to the agent log file.

3. Use the OEPROP utility to specify deferred log entry type:

```
oeprop.{bat|sh} AppServer.SessMgr.defrdLogEntryTypes=4GLTrace
```

   The default is the `4GLTrace` log entry type. The following agent log entry types are supported for deferred logging:

   - `4GLTrace`
   - `ASPlumbing`
   - `UBNET`
   - `DB.Connects`

   Note: Deferred logging does not currently support all log entry types that can be configured for the PAS for OpenEdge agent. If deferred logging is turned on, and an unsupported log entry type is specified in the `deferredLogEntryTypes` property, log messages for the unsupported log entry type will be written to the agent log file. At present, `DYNOBJECT` is not supported for deferred logging.

4. When a severe error occurs on a multi-session agent, a graceful emergency shutdown is initiated to avoid further failures. Along with updating the Syslog (UNIX) or Eventslog (Windows) file and generating the Protrace, the shutdown process also creates a new file, `pasoe-agent-crash-trace-`*`process-id`*. For more information, see Graceful emergency shutdown of a multi-session agent on page 292.

   Alternatively, you can invoke an API to print the deferred logging buffer to the agent log file or clear the deferred log buffer. For more information, see:

   - REST APIs: "Flush deferred log buffer to agent log file" and "Reset deferred log buffer" in *PAS for OpenEdge Administration Tools Reference*.
   - OEJMX queries: "Flush deferred log buffer to agent log file query" and "Reset deferred log buffer query" in *PAS for OpenEdge Administration Tools Reference*.

Use OEJMX to manage and monitor an instance

You can monitor and manage applications using the Java Monitoring and Management Console (JConsole) and the OpenEdge Manager application (`oemanager`). You can also monitor and manage instances through scripts using the OpenEdge JMX command-line utility, OEJMX, to connect to a local PAS for OpenEdge instance.

Based on Java Management Extensions (JMX), the `OEJMX` utility connects to an instance using Java objects called Managed Beans (MBeans). One or more MBeans instrument a resource using attributes and operations. This direct access to the underlying MBeans provides administrators with the freedom to write scripts to monitor and manage applications in ways that may not be available within OpenEdge Manager or JConsole.

For the OEJMX command syntax, see OEJMX utility.

For usage information, see the following topics:

- List available MBeans to locate AgentManager details on page 244
- Use an OEJMX query to get agent information on page 246
- Use an OEJMX query to get session metrics on page 248
- Common OEJMX queries on page 250

List available MBeans to locate AgentManager details

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 on a running PAS for OpenEdge instance with the `-C` parameter to generate a list of the available MBeans and related operations and attributes. By default, the output is sent to the *instance-name*`/temp/beanInfo-`*timestamp*`.txt` file. With a complete list of MBeans, you 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:

1. From the target PAS for OpenEdge instance's `bin` directory, start the instance, for example:

```
proenv>tcman pasoestart
```

2. Run `oejmx -C` to create a list of available MBeans, for example:

```
proenv>oejmx -C
```

   Output is sent to the default location,`/temp/beanInfo-`*timestamp*`.txt`.

3. Open the output file and search for the `AgentManager` details. The AgentManager MBean lists the available operations that can be used to get agent information, status, and other metrics. The JMX queries are displayed on the lines that begin with `{"O":"PASOE:type=OEManager,name=AgentManager"` as shown in the sample AgentManager MBean operations output:

```
**************** PASOE:type=OEManager,name=AgentManager
Operation: getAgents (String appName) -->  javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getAgents","<appName>"]}
Operation: refreshAgents (String appName) -->  javax.management.openmbean.CompositeData
```

```
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["refreshAgents","<appName>"]}
Operation: refreshAgent (String id) -->  javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["refreshAgent","<id>"]}
Operation: getAgentStatus (String id) -->  javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getAgentStatus","<id>"]}
Operation: flushDeferredLog (String appName, String agentId) -->
javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["flushDeferredLog","<appName>","<agentId>"]}
Operation: resetDeferredLog (String appName, String agentId) -->
javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["resetDeferredLog","<appName>","<agentId>"]}
Operation: clearAgentStatData (String id) -->  Boolean
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["clearAgentStatData","<id>"]}
Operation: getAgentMetrics (String id) -->  javax.management.openmbean.CompositeData

  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getAgentMetrics","<id>"]}
Operation: getThreadMetrics (String id) -->  javax.management.openmbean.CompositeData

  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getThreadMetrics","<id>"]}
Operation: getSessionMetrics (String id) -->  javax.management.openmbean.CompositeData

  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getSessionMetrics","<id>"]}
Operation: getConnectionMetrics (String id) -->
javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getConnectionMetrics","<id>"]}
Operation: getRequestMetrics (String id) -->  javax.management.openmbean.CompositeData

  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getRequestMetrics","<id>"]}
Operation: getAgentStacks (String id) -->  javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getAgentStacks","<id>"]}
Operation: getSessionStacks (String agentID, String sessionID) -->
javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["getSessionStacks","<agentID>","<sessionID>"]}
Operation: stopAgent (String agentID, Integer waitToFinish, Integer waitAfterStop)
-->  boolean

{"O":"PASOE:type=OEManager,name=AgentManager","M":["stopAgent","<agentID>",<waitToFinish>,<waitAfterStop>]}
Operation: getDynamicABLSessionLimit (String appName, String id) -->
javax.management.openmbean.CompositeData[]

{"O":"PASOE:type=OEManager,name=AgentManager","M":["getDynamicABLSessionLimit","<appName>","<id>"]}
Operation: resetDynamicABLSessionLimit (String appName, String id) -->
javax.management.openmbean.CompositeData[]

{"O":"PASOE:type=OEManager,name=AgentManager","M":["resetDynamicABLSessionLimit","<appName>","<id>"]}
Operation: setDynamicABLSessionLimit (String appName, String id, Integer sessionLimit)
 -->  javax.management.openmbean.CompositeData[]

{"O":"PASOE:type=OEManager,name=AgentManager","M":["setDynamicABLSessionLimit","<appName>","<id>",<sessionLimit>]}
Operation: terminateABLSession (String agentID, Integer agentSessionID, Integer
terminateOption) -->  boolean

{"O":"PASOE:type=OEManager,name=AgentManager","M":["terminateABLSession","<agentID>",<agentSessionID>,<terminateOption>]}
Operation: trackABLObjects (String agentID, Boolean switch) -->  Boolean

{"O":"PASOE:type=OEManager,name=AgentManager","M":["trackABLObjects","<agentID>",<switch>]}
Operation: trackingABLObjects (String agentID) -->  boolean
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["trackingABLObjects","<agentID>"]}
Operation: getABLObjectsReport (String agentID) -->
javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["getABLObjectsReport","<agentID>"]}
Operation: getABLSessionObjectsReport (String agentID, Integer agentSessionID) -->
 javax.management.openmbean.CompositeData
```

```
{"O":"PASOE:type=OEManager,name=AgentManager","M":["getABLSessionObjectsReport","<agentID>",<agentSessionID>]}
Operation: trackMemoryUse (String agentID, Boolean switch) -->  Boolean

{"O":"PASOE:type=OEManager,name=AgentManager","M":["trackMemoryUse","<agentID>",<switch>]}
Operation: trackingMemoryUse (String agentID) -->  boolean
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["trackingMemoryUse","<agentID>"]}
Operation: getMemoryUseReport (String agentID) -->
javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getMemoryUseReport","<agentID>"]}
Operation: getMemoryUseSessionReport (String agentID, Integer agentSessionID) -->
javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["getMemoryUseSessionReport","<agentID>",<agentSessionID>]}
Operation: initiateDebugger (String id, String connectString) -->  boolean

{"O":"PASOE:type=OEManager,name=AgentManager","M":["initiateDebugger","<id>","<connectString>"]}
Operation: debugTest (String agentID, String operation, String modifiers, Integer
iparm, String cparm) -->  javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["debugTest","<agentID>","<operation>","<modifiers>",<iparm>,"<cparm>"]}
Operation: pushProfilerData (String id, String url, Integer requestCount, String
profilerSettings) -->  javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["pushProfilerData","<id>","<url>",<requestCount>,"<profilerSettings>"]}
Operation: getProfilerSettings (String id) -->
javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getProfilerSettings","<id>"]}
Operation: dynamicPropertiesUpdate (String id, String propertiesString) -->  boolean


{"O":"PASOE:type=OEManager,name=AgentManager","M":["dynamicPropertiesUpdate","<id>","<propertiesString>"]}
Operation: getProperties (String appName) -->  javax.management.openmbean.CompositeData

  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getProperties","<appName>"]}
Operation: setProperties (String appName, String PropertyName, String PropertyValue)
 -->  javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["setProperties","<appName>","<PropertyName>","<PropertyValue>"]}
```

For example, the `stopAgent` operation could be used to stop an agent, and the `terminateABLSession` operation could be used to stop a particular session. These operations are described in the OEJMX Query Reference.

Some JMX queries require you to replace the variables with information obtained by running a `getAgents` query, or by providing other data such as application name, property name, or URL. Writing the `getAgents` query is described in Use an OEJMX query to get agent information on page 246.

Note: Detailed information about available MBeans can be found by using JConsole. For more information, see JConsole and JMX.

With a list of available MBeans, you can Use an OEJMX query to get agent information on page 246.

Use an OEJMX query to get agent information

Use OEJMX with the `-Q` *`filepathname`* parameter to use a custom query file. A query file can have multiple queries. For more information about the OEJMX command syntax, see OEJMX utility on page 43.

Each JMX query is a JSON string with an MBean identifier and attribute or operation information. The query format is:

- **0**—ObjectName is a JsonObject

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

An example is the JMX query for getting agent information:

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

In List available MBeans to locate AgentManager details on page 244, steps are provided to list the available MBeans. 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 the `getAgents` API for the default `oepas1` instance.

To write a custom OEJMX query:

1. Create a new file in your PAS for OpenEdge instance's `bin\jmxqueries` directory called `getAgents.qry`.
2. Add the following line to the `getAgents.qry` file:

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

   This line contains a JMX query to access the **AgentManager** MBean, and specifically the `getAgents` JMX API. This provides information about the running agents in the PAS for OpenEdge instance, which can be used to get thread and session information, status, and other metrics.

   The query returns the agent information for the default `oepas1` instance. If you are running this query on a PAS for OpenEdge instance with a different name, change `oepas1` to the name of the current instance or ABL application that you want to get agent information about.

   If you want agent information for all running ABL applications on the PAS for OpenEdge instance, you can specify `APP_NAME` instead of `oepas1` (or other ABL application name). `APP_NAME` is a supported token that returns information for all ABL applications on the PAS for OpenEdge instance. More information about supported tokens is described inCommon OEJMX queries on page 250.

   Note: A query must be a single line that does not contain 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.

3. Navigate to the `bin` directory of the instance, for example:

   ```
   proenv>cd %WRKDIR%\oepas1\bin
   ```

4. Ensure the instance is running by executing the following command:

   ```
   proenv>tcman pasoestart -restart
   ```

5. Run the OEJMX getAgents query that you created in step one, for example:

   ```
   proenv>oejmx -Q jmxqueries\getAgents.qry
   ```

The default output location is *instance_name*/temp/*queryname*-out-*yyMMdd-HHmm*.txt.

6. Review the output getAgents-out*yyMMdd-HHmm*.txt file in the *instance_name*/temp directory to see the query response, for example:

```
Query: Line 1. PASOE:type=OEManager,name=AgentManager, getAgents(oepas1)
Result:
{"getAgents":{"agents":[{"agentId":"b4XlcFFOR_uh68bStvHe1Q","pid":"26896","state":"AVAILABLE"}]}}
```

Note: The output shows the original query on line 1 and the results on the next line. The original query message can be suppressed by running the OEJMX query with the -R option, for example:

```
proenv>oejmx -Q jmxqueries\getAgents.qry -R
```

For more information about the OEJMX command syntax, see OEJMX utility on page 43.

The output of the getAgents query provides the Agent ID and PID for the running ABL application agents. You can use this information to gather more information about your agent, such as threads and session metrics. For more information, see Use an OEJMX query to get session metrics on page 248 and Common OEJMX queries.

Use an OEJMX query to get session metrics

OEJMX provides access to agent and session metrics, which can give administrators insights into the PAS for OpenEdge instance. This topic discusses how to generate session metrics for an agent, and shows the general process for running these types of queries. There are numerous other JMX APIs available which are described in Common OEJMX queries on page 250 and the OEJMX Query Reference. These queries can provide different levels of insight into a PAS for OpenEdge instance.

Additionally, the APIs described in the AgentManager section of the oejmx -C output are available for use as well. These operations are described in the List available MBeans to locate AgentManager details on page 244 topic.

To use OEJMX to generate session metrics for an agent:

1. Ensure the PAS for OpenEdge instance is running.
2. Obtain the Agent ID or PID for the agent that you want to query for session metrics. These can be obtained by running the getAgents query, which is described in Use an OEJMX query to get agent information on page 246.

   Note: In place of the Agent ID or PID, you can use the `AGENT_ID` token, which provides information for all running agents. However, the best practice is to use explicit names over tokens to guarantee the intended results.

3. In the *instance_name*`\bin\jmxqueries` directory, create a file called `getSessionMetrics.qry`, and insert the following line into the file:

   ```
   {"O":"PASOE:type=OEManager,name=AgentManager","M":["getSessionMetrics","31288"]}
   ```

   In the above example, replace *31288* with the Agent ID or PID that you obtained for a running agent.

   Note: To obtain session metrics for each running agent on the PAS for OpenEdge instance, specify the `AGENT_ID` token in place of the Agent ID value, for example:

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

   For more information about tokens, see OEJMX utility on page 43.

4. Run the getSessionMetrics query using the OEJMX command from the bin directory of the instance, for example:

   ```
   proenv>oejmx -Q jmxqueries\getSessionMetrics.qry
   ```

   The default output location is *instance_name*`/temp/getSessionMetrics-out-`*yyMMdd-HHmm*`.txt`.

5. Review the output `getSessionMetrics-out`*yyMMdd-HHmm*`.txt` file in the *instance_name*`/temp` directory to see the query response. It looks similar to the following example:

   ```
   Query: Line 1. PASOE:type=OEManager,name=AgentManager,
   getSessionMetrics(b4XlcFFOR_uh68bStvHe1Q)
   Result:
   {"getSessionMetrics":{"AgentSession":[{"SessionState":"IDLE","ConnectionId":"org.codehaus.jettison.json.JSONObject$Null",
   "SessionMemory":758020,"EndTime":"org.codehaus.jettison.json.JSONObject$Null",
   "ThreadId":-1,"SessionExternalState":0,"StartTime":"2020-09-10T11:32:44.485+04:00","SessionId":4},
   {"SessionState":"IDLE","ConnectionId":"org.codehaus.jettison.json.JSONObject$Null",
   "SessionMemory":758020,"EndTime":"org.codehaus.jettison.json.JSONObject$Null",
   "ThreadId":-1,"SessionExternalState":0,"StartTime":"2020-09-10T11:32:44.485+04:00","SessionId":7}]}}
   ```

   The output of the getSessionMetrics query provides the session IDs for the running ABL application agents. You can use this information to gather more information about your agent, such as generating a stack trace. For more information about generating a stack trace, see the Get a stack trace for a session query topic.

This topic shows the basic procedure for writing OEJMX queries, running them on the console, and viewing the output of the query. Multiple queries can be included in a single OEJMX query file. More examples of OEJMX queries can be found in Common OEJMX queries on page 250 and the OEJMX Query Reference.

Common OEJMX queries

One of the fundamental OEJMX queries is to request agents for an ABL application. This query provides you with information about the multi-session agents running in your application. You can add these queries to build out a multi-line query file to get information about your PAS for OpenEdge instance.

Get agents from an ABL application

When to use: To identify agents in the ABL application.

Query:

```
# Tokens: APP_NAME
# Get agents in an ABL application
{"O":"PASOE:type=OEManager,name=AgentManager","M":["getAgents","oepas1"]}
```

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

Result:

The output provides details about the agents including agent ID, PID, and state information. The agent ID can be used in other OEJMX queries such as to refresh an agent in an ABL application.

```
{
  "getAgents": {
    "agents": [
      {
        "agentId": "LMufO6dzRiSguShJOsJ53Q",
        "pid": "13992",
        "state": "AVAILABLE"
      }
    ]
  }
}
```

Additional queries

Additional OEJMX queries can be found in "OEJMX Query Reference" in *PAS for OpenEdge Administration Tools Reference*, including:

- Get agents query
- Get active session information query
- Get a stack trace for a session query
- Get metrics for an ABL application query
- Get hung client connections for an ABL application query
- Get agent metrics query
- Get agent session metrics query
- Get active threads from Tomcat thread pool query

- Refresh agent query
- Refresh all agents query
- Flush deferred log buffer to agent log file query
- Reset deferred log buffer query
- Get dynamic ABL session limit query
- Update dynamic ABL session limit
- Reset dynamic ABL session limit query

Use Server-Side ABL Performance Profiling

PAS for OpenEdge system administrators and developers can use Server-Side ABL Performance Profiling to gather and monitor run-time performance data about ABL applications running on a PAS for OpenEdge instance.

This topic explains how to configure a PAS for OpenEdge instance to use performance profiling, configure the diagnostic store application, and view performance data using the Profiler tool in Progress Developer Studio.

Server-Side ABL Performance Profiling is referred to as Server-Side Profiling for brevity in this document.

See also:

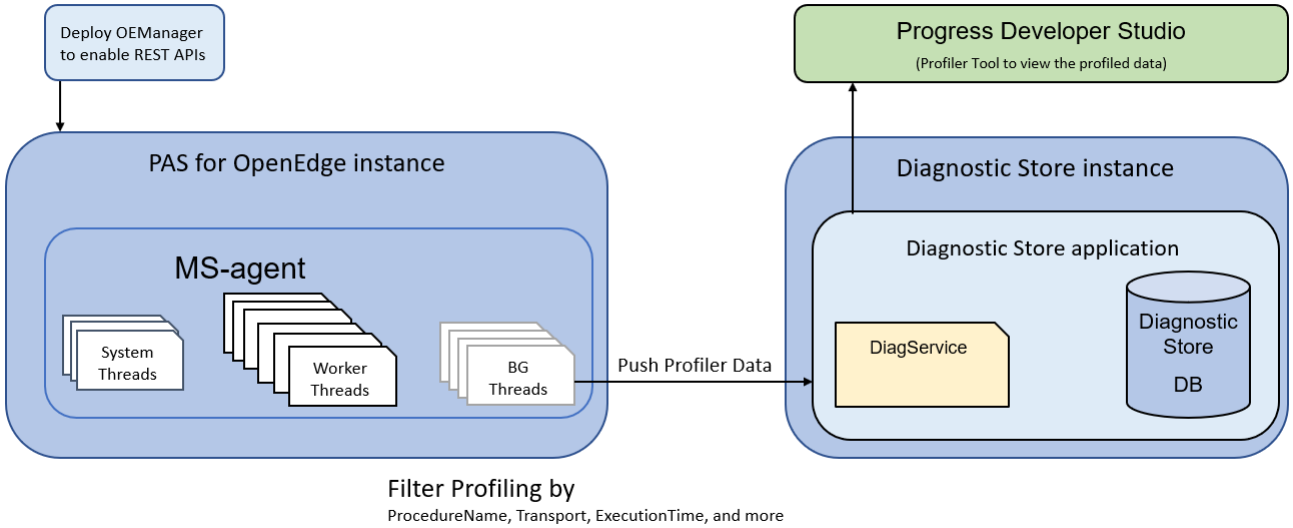
- What is Server-Side Profiling?
- Configure the diagnostic store
- Configure Server-Side Profiling with PAS for OpenEdge
- Supported JSON values for JMX queries

What is Server-Side Profiling?

Use PAS for OpenEdge Server-Side Profiling to gather and monitor run-time performance data about ABL applications running on a PAS for OpenEdge instance. Performance data can be obtained for both development and production instances of PAS for OpenEdge. Instance shutdown is not required to configure an instance for profiling and enable an instance to gather data.

The following figure shows a Server-Side ABL Performance Profiling workflow and architecture:

Figure 3: Server-Side ABL Performance Profiling architectural diagram



When Server-Side Profiling is enabled for a PAS for OpenEdge instance:

1. One or more multi-session agents gather performance data.
2. The performance data is pushed to a diagnostic store.
3. View the data using a profiler tool such as Progress Developer Studio.

Progress Software provides a default diagnostic store web application that you can deploy to an instance. The diagnostic store is located on a PAS for OpenEdge instance called Diagnostic Store; however, you can use your own implementation of a web server as a diagnostic store. You can then use a profiler tool such as Progress Developer Studio to view the profiler data.

The main steps for configuring Server-Side Profiling are:

1. Configure the diagnostic store
2. Configure Server-Side Profiling with PAS for OpenEdge

Configure the diagnostic store

The diagnostic store is a web application deployed on a PAS for OpenEdge instance that receives performance data from deployed ABL applications, and stores that data in an OpenEdge database. You can either create a new PAS for OpenEdge instance for the diagnostic store or use an existing instance.

To configure the diagnostic store:

1. Create an instance to host the diagnostic store application.

   To create a new PAS for OpenEdge instance, use the PASMAN command-line utility, and specify the ports as well as the instance name:

   ```
   pasman create -p 8810 -P 8811 -s 8812 diagstore
   ```

This command creates an instance named `diagstore`.

2. Deploy the diagnostic store web application to `diagstore`:

```
pasman deploy -I diagstore $DLC/servers/pasoe/extras/oediagstore.war
```

3. Create a database for the diagnostic store called `oediagdb`:

```
cd diagstore/work
prodb oediagdb empty
```

4. Load the database definitions file to `oediagdb` database:

   a. Copy `$DLC/servers/pasoe/extras/oediagstore.df` to the current directory:

   ```
   [copy|cp] $DLC/servers/pasoe/extras/oediagstore.df diagstore/work
   ```

   b. Start the database:

   ```
   proserve oediagdb -S 5555
   ```

   c. Open the Procedure Editor with the `oediagdb` database parameter:

   - On UNIX:

     ```
     _progres -db oediagdb
     ```

   - On Windows:

     ```
     prowin -db oediagdb
     ```

   d. Run the following ABL code from the Procedure Editor:

   ```
   RUN prodict/load_df.p (INPUT "oediagstore.df").
   QUIT.
   ```

5. Verify that the diagnostic store instance starts successfully:

```
pasman pasoestart -I diagstore -timeout 200
```

Configure Server-Side Profiling with PAS for OpenEdge

This topic explains how to use JMX to enable Server-Side Profiling on a PAS for OpenEdge instance. You may not want to gather performance data for every request handled by an instance. Use filters to restrict which requests are profiled on a particular instance. Filters are configured and enabled using JMX and REST APIs. Filters are described in Supported JSON values for JMX queries.

Before performing the steps in this topic, ensure that the diagnostic store is deployed and running.

Note: Line breaks are not supported in JMX queries ran through the OEJMX utility. The queries must be written on a single line, even though they appear formatted in this topic.

In this procedure, you configure profiling on the target PAS for OpenEdge instance and send the profiled data to the diagnostic store:

1. Ensure that the diagnostic store and the target PAS for OpenEdge instance are running. The target instance requires the `oemanager.war` application is deployed to access REST APIs.
2. Run an JMX query to return the list of MS-agents running for a given ABL application on the target instance:
   a. In the `/bin` directory of the target instance, create a JMX query called `agentInfo.qry`:

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

      APP_NAME is the ABL application you want to profile.

   b. Run the query:

      ```
      oejmx.[bat|sh] -R -Q agentInfo.qry
      ```

      The output of the file is sent to the *instance-name*`/temp` directory

   c. Open the `agentInfo` output file, and copy the `agentId` value. This ID is used in the following steps.

3. In the `/bin` directory of the target instance, create a JMX query to capture diagnostic data.

   The following example query captures diagnostic information for the next 100 server status requests. It also uses the `AdapterMask` and `Proclist` filters. See Supported JSON values for JMX queries for a list of supported Profiler settings.

   ```
   {"O":"PASOE:type=OEManager,name=AgentManager",
   "M":["pushProfilerData","AGENT_ID",
   "http://host:port/oediagstore/web/diag",100,
   "{\"Coverage\":true,\"Statistics\":true,\"AdapterMask\":\"REST\",
   \"ProcList\":\"OpenEdge.Rest.Admin.AppServerStatus:ServerStatus\",
   \"TestRunDescriptor\":\"Sample Test\"}"]}
   ```

   - *host* and *port* point to the diagnostic store.
   - *AGENT_ID* is the agent ID running on the target instance.

4. Execute the query to start profiling:

   ```
   oejmx.[bat|sh] -R -Q diaginfo.qry
   ```

   The output is sent to the *instance-name*`/temp` directory.

   The following output is returned:

   ```
   {"pushProfilerData":
   {"ABLReturnVal":true,
   "agentId":"solo8Wm7TquALsSQ1lpX4g",
   "pid":"8116"}}
   ```

   If the result of `ABLReturnVal` is `true`, the JMX query executed successfully, and profiling is enabled on the target PAS for OpenEdge instance.

5. Execute the following query to get the profiler settings and save it in the *instance-name*`/bin` directory as `profilersettings.qry`:

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

   `AGENT_ID` is the agent ID.

6. Run the following query to show that the profiler settings for the target PAS for OpenEdge instance are enabled:

   ```
   oejmx.[sh|bat] -R -Q profilersettings.qry
   ```

   The output is sent to the target *instance-name*/temp directory.

   The following result is returned:

   ```
   {"getProfilerSettings":{"ABLOutput":{"AdapterMask":"REST",
   "ProcList":"OpenEdge.Rest.Admin.AppServerStatus:ServerStatus",
   "RequestHiBound":-1,"TestRunDescriptor":"Sample Test",
   "bufsize":128,"RequestLoBound":0,
   "Coverage":true,"RequestCount":100,"Statistics":true,
   "URL":"http://172.29.36.15:8810/oediagstore/web/diag"},
   "ABLReturnVal":true,"agentId":"solo8Wm7TquALsSQ1lpX4g","pid":"8116"}}
   ```

7. Run a sample _oeping service on the target instance to see if the REST service ABL code is profiled and sent to the diagnostic store.

   If the request matches the filters you specified in the JMX query, then the agent in the instance pushes the profiling data to the diagnostic store.

   ```
   curl http://host:port/rest/_oepingService/_oeping
   ```

   The _oepingService calls the ABL class OpenEdge.Rest.Admin.AppServerStatus and invokes ServerStatus. Because the server status matches the filter, and the AdapterMask is REST, this request is profiled and the profiler output us sent to the diagnostic store. For more information about the OpenEdge PING service, see ABL application PING service.

8. To view and analyze the profiler data, export the data from the diagnostic store (oediagdb) to a .prof file, and open that file in Progress Developer Studio.

   To export the data, run the following ABL code after connecting to the diagnostic store database:

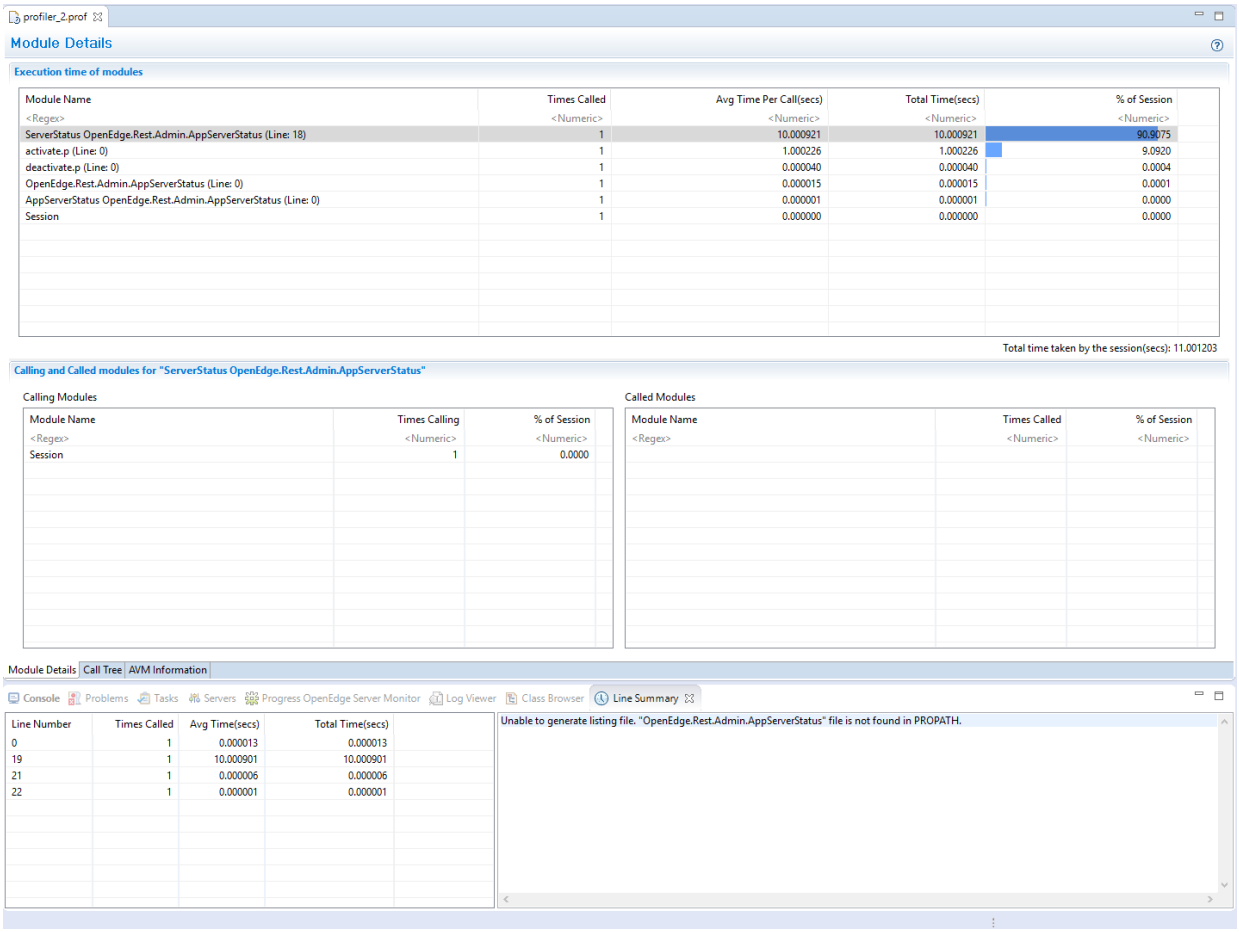
   ```
   DEFINE VAR requestId as Integer NO-UNDO.
   DEFINE VAR fileName as Character NO-UNDO.
   FOR EACH ProfiledRequest:
    requestId = requestId + 1.
    fileName =  "C:/OpenEdge/WRK/profiler_" + STRING(requestId) + ".prof".
       COPY-LOB ProfiledRequest.PerfData TO FILE fileName.
       MESSAGE fileName VIEW-AS ALERT-BOX.
   END.
   ```

   Note: On UNIX or Windows, replace C:/OpenEdge/WRK with the full system path to the WRK directory where you want to put the profiler files.

9. From the specified WRK directory, open the generated .prof file in Progress Developer Studio, or another profiler tool, to view the profiled data.

In Progress Developer Studio, the profiled data appears in the Module Details page. It shows a summary of the events observed during the run-time ping request.

Figure 4: Example ABL profiler data in Progress Developer Studio



Supported JSON values for JMX queries

You can write custom JMX queries for profiling data based on specific filters available as JSON values.

The supported input JSON values for JMX queries are as follows:

- (Required) Target URL where the data is to be pushed
- (Required) Number of requests to push data for:
  - `0` means no requests
  - `-1` means no limit and continue until reset
- Profiler settings as a JSON object. This object can have one or more of the following optional values. If the value is not specified, then the settings keep their default values. These settings act as filters for capturing profiled data:

Table 15: Supported Profiler Settings

Value	Type	Description	Default
`Coverage`	Boolean	Sets ABL `PROFILER:COVERAGE` on or off.	`False`
`Statistics`	Boolean	Sets ABL `PROFILER:STATISTICS` on or off.	`False`
`TestRunDescriptor`	String	Identifier for the test run, ABL `PROFILER:DESCRIPTION`	`""`
`bufsize`	Integer	Size of buffer in KB for logging profiler data. Minimum: 128; Maximum: 1024.	`128`
`ProcList`	String	A CSV filter of procedures to be profiled. An empty string means profile all.	`""`
`AdapterMask`	String	A CSV filter of adapters. Supported adapters: REST, APSV, SOAP, and WEB. `nil` means accept all.	`""`
`RequestLoBound`	Integer	Filter for minimum request length to be profiled in milliseconds.	`0`
`RequestHiBound`	Integer	Filter for minimum request length to be profiled in milliseconds. `-1` means no limit.	`-1`

Use Swagger UI to explore management REST APIs

System administrators can use Swagger UI to explore the underlying management APIs for a running PAS for OpenEdge instance. Part of the larger OpenAPI Initiative, Swagger UI is an open source tool that generates an intuitive web page to explore and test REST APIs. Details are pulled directly from the underlying annotations ensuring that the documentation is always current. Test capabilities allow real-time access to the APIs. System administrators can view metrics, identify deployed applications, stop agents, and do much more through this interactive documentation.

Access to Swagger UI is disabled by default, to prevent unauthorized users from tampering with an instance.

In OpenEdge 12.0 and 12.1, Swagger UI is disabled by default on development instances. In OpenEdge 12.2, Swagger UI is enabled by default on development instances. Swagger UI should not be used in production.

See also:

- Enable Swagger UI for management REST API access on page 258
- Explore and test management REST APIs on page 259

Enable Swagger UI for management REST API access

Management REST API access requires `oemanager.war` be deployed. Create development instances with the `-f` option to deploy `oemanager.war`. Secure the instance by using the `-m` *`uid:password`* option to change the username and password from the default values. Start the instance at least once to create the `oemanager` folder structure containing the necessary configuration files.

Note: This topic applies to Swagger UI in OpenEdge 12.0 and 12.1. In OpenEdge 12.2, Swagger UI is enabled by default on development instances. Swagger UI should never be enabled in production.

Access to Swagger UI for management REST APIs is disabled by default. Direct access to management APIs through Swagger UI is considered a security risk. Explicit authorization is required. Limit access to the administrator roles within the organization. Restart the server to apply the security changes. To enable Swagger UI:

1. Edit the *`instance-path`*`\webapps\oemanager\WEB-INF\oemgrSecurity-container.xml` security file.
2. Find the intercept URL for the interactive documentation, by searching the file for `<intercept-url pattern="/doc/**"`.
3. Uncomment the line containing `<intercept-url pattern="/doc/**` that authorizes specific roles to use the interactive documentation.
4. Save the file.
5. Restart the server to apply the changes:

   ```
   pasman pasoestart -restart -I instance-name
   ```

6. Enter `http://`*`host`*`:`*`port`*`/oemanager/` to show the management REST API documentation, confirming that authorization was correctly defined. Login credentials are required to ensure only administrators have access.

Explore and test management REST APIs

Access to Swagger UI is disabled by default. To enable this functionality, see Enable Swagger UI for management REST API access on page 258.

In this example, explore the **SessionManager** APIs and get all the available applications for an instance. Based on the response, use the **Agent Manager** getAgent API to get the PID for the agent currently running a named application. To explore and test a REST API:

1. Enter `http://host:port/oemanager/` in a browser to access interactive documentation, and provide the appropriate login credentials when prompted.
2. Verify that `oemanager` is the selected **Server**.
3. Scroll to **SessionManager** and expand the list of available APIs.
4. Click on **GET** for the `/applications - Get All Applications` to expand the description based on the underlying annotations.
5. Click **Try it out** to supply request parameters and click **Execute** to retrieve a list of available applications.

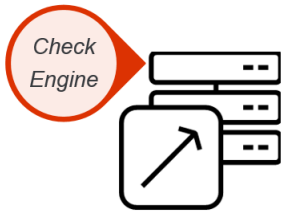
   The response body is JSON formatted and the `applicationName` element provids details about the deployed applications, including which transports are enabled for the application and the version information.
6. Take note of the `applicationName` for use as the request body to find the Agent running the application.
7. Expand the **Agent Manager** section.
8. Click **/applications/{appName}/agents Get Agents**.
9. Click **Try It Out**.
10. Enter the `applicationName` returned in the earlier step into the Parameters **AppName** field.
11. Click **Execute**.

    The response returns a **PID**, for use with API requests.

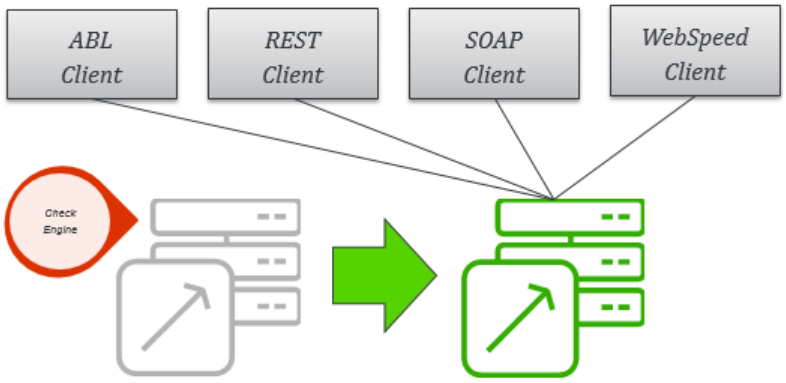
For more information, see Monitor PAS for OpenEdge.

About the OpenEdge HealthScanner

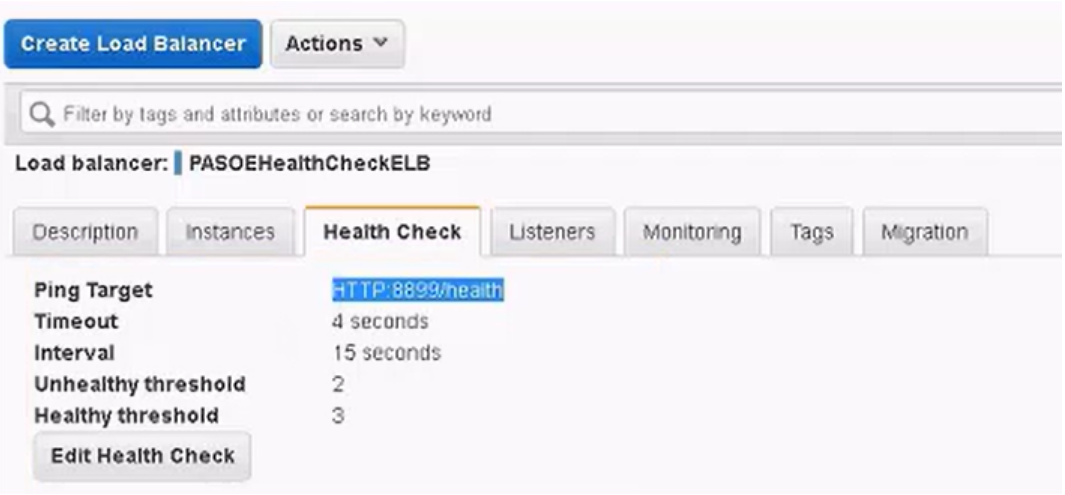
Like the check engine light for a vehicle, the OpenEdge HealthScanner alerts system administrators of potential problems with an instance so that the server can be taken out of service before a failure occurs. The OpenEdge HealthScanner is not designed to research or resolve the problem; it flags a server that falls below a healthy threshold.



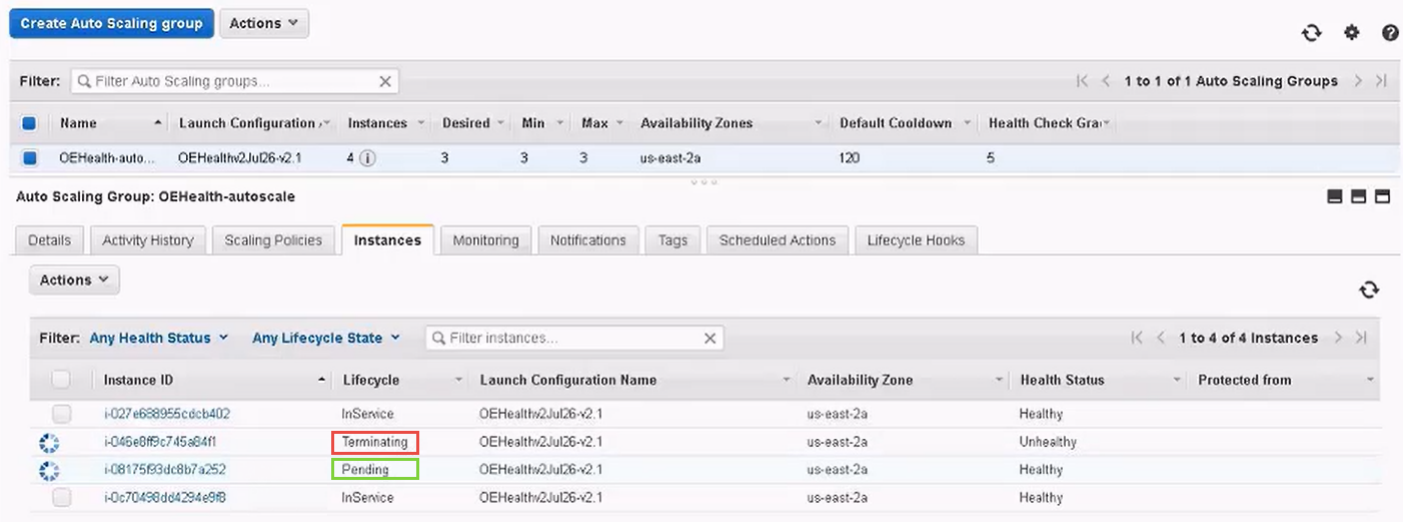
Designed for system administrators, the OpenEdge HealthScanner checks the overall health of a server, and assigns a score from 0-100% as well as a status code. System administrators configure an acceptable threshold for a server to be considered in operational good health. Servers with scores below the threshold can be taken out of service by an elastic load balancer (ELB). The ELB can terminate the server, and can dynamically replace the failing server before any disruption of service happens.



The following is an example of the Amazon Web Service (AWS) Cloud Health Check configuration for a server that is part of an Auto Scaling group. Use the AWS Health Check **ping target** field to poll the health of a server.



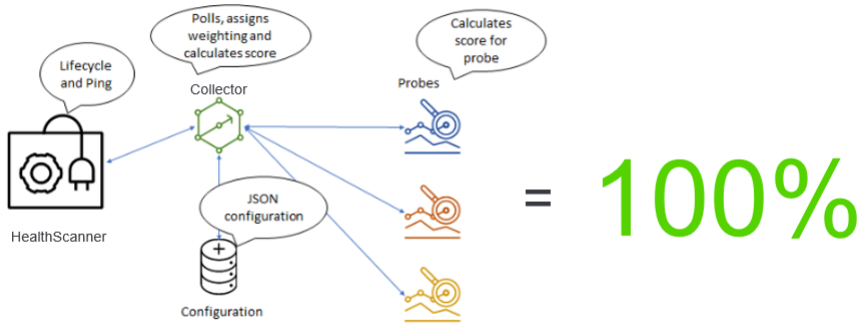
If the ping returns a status code of 500, then the server is automatically terminated, and the new server is brought online, providing continuous availability of a healthy server to respond to client requests.



Note: The HealthScanner is limited to cloud operating systems for Windows and Linux.

How is the data collected?

The HealthScanner consists of a health collector that collects data from a variety of probes. Each probe checks a specific health value of the server, just like the sensors in a vehicle checking the engine, oil, gas, and tire pressure levels.



The HealthScanner is a Tomcat service with its own HTTP port, thread pool, and configuration. As a stand-alone service, it returns probe results even when the main Tomcat connector is not able to respond. This feature is an opt-in service, to avoid unnecessary use of system resources.

Use the OpenEdge HealthScanner

The OpenEdge HealthScanner is intended for use with production instances. System administrators must enable the IP filter to secure the connection used by the HealthScanner.

As an opt-in feature, system administrations must enable the HealthScanner and enable the overhead associated with data collection. To simplify the demonstration of this feature, a development instance is used.

Enable the HealthScanner

The OpenEdge HealthScanner is disabled by default. Use the `tcman feature` command to enable it.

1. Create a PAS for OpenEdge development instance in the OpenEdge WRK directory, for example:

```
proenv>cd %WRKDIR%
proenv>pasman create -Z dev HealthInst
```

Note: The `-Z dev` option indicates a development security model, rather than a production security configuration.

2. Change to the `bin` directory of the instance:

```
proenv>cd instance-name\bin
```

3. List the *feature*=*value* pairs using the `tcman feature` command:

```
proenv>tcman feature
VersionLoggerListener=off
SecurityListener=off
APRListener=off
PSClifecycle=on
JMXLifecycle=off
PSCRegistry=on
HealthCheck=off
HTTP=on
HTTPS=on
AJP13=off
Cluster=off
UserDatabase=on
JAASRealm=off
LDAPRealm=off
PASInstrument=off
RemoteHostValve=on
RemoteAddrValve=on
SingleSignOn=on
AccessLog=on
CrawlerSessionManager=on
StuckSessionValve=off
```

By default, the feature list indicates that `HealthCheck=off`, which means that the feature is disabled.

4. Enable the HealthCheck feature with the following command:

```
proenv>tcman feature HealthCheck=on
```

5. To verify the change, use TCMAN to poll the HealthCheck value:

```
proenv>tcman feature HealthCheck
HealthCheck=on
```

Turn on data collection

The data collector for the OpenEdge HealthScanner must be enabled in addition to the HealthCheck feature. The settings to enable and configure the data collector for the OpenEdge HealthScanner are located in the *instance-name*\conf\appserver.properties file.

When the data collector is enabled, it reads the results from the probes which are documented in the health.config file.

To enable data collection for the OpenEdge HealthScanner, perform the following steps:

1. Enable the collector to collect data from the probes using the tcman config command:

```
proenv>tcman config psc.as.health.enabled=true
```

Note: To echo back the newly set value, enter the following command:

```
proenv>tcman config psc.as.health.enabled
```

2. To learn more about the probes that provide data to the OpenEdge HealthScanner, see the health.config file located in the *instance-name*\conf directory.
3. Restart the server:

```
proenv>tcman pasoestart -restart
```

4. Confirm that the HealthScanner is running:

```
proenv>tcman env
proenv>tcman env
catalina home:     C:\Progress\OpenEdge\servers\pasoe
catalina base:     C:\OpenEdge\WRK\HealthInst
catalina tmpdir    C:\OpenEdge\WRK\HealthInst\temp
catalina pid:      C:\OpenEdge\WRK\HealthInst\logs\catalina-HealthInst.pid
java home:         C:\Progress\OpenEdge\jdk
jre home:
manager http port: 8810
manager https port:8811
manager shut port: 8812
manager ajp13 port: 8009
manager health port: 8899

manager URL:       http://localhost:8810/manager
config type:       instance
config alias:      HealthInst
config parent:     C:/Progress/OpenEdge/servers/pasoe
server running:    1
instance tracking: True
instance file:     C:\Progress\OpenEdge\servers\pasoe\conf\instances.windows
server process-id: 28040
window title:      PAS-HealthInst-54629
security model:    developer
service:           false
servicename:
```

The line manager health port: 8899 identifies the port for the HealthScanner.

You are ready to Access HealthScanner data on page 264, which can be displayed in a variety of formats and in different configurations.

Access HealthScanner data

The following table list ways to request data returned by the HealthScanner probes for a running PAS for OpenEdge instance:

Request	Return value
http://hostname:8899/health	Returns a status code. Typically, `200` when healthy, and `500` if the health falls below a threshold. For more information, see Return the HealthScanner status code.
http://hostname:8899/health?view=summary	Returns a JSON formatted summary of the health score. For more information, see Return a JSON summary view.
http://hostname:8899/health?view=details	Returns a JSON formatted details of the health probes. For more information, see Return a JSON details view.
http://hostname:8899/health?view=config	Provides a dataset-friendly view of the HealthScanner probe configuration.
http://hostname:8899/health?view=dataset	Provides a dataset-friendly view of HealthScanner probe results.
instance-name/bin/oehealth	Uses OEJMX to gather the health values. This information could be pushed to a raw data collector for use with administrative scripts. For more information, see Return as an OEJMX query.

Additionally, you can Write a custom HealthScanner view.

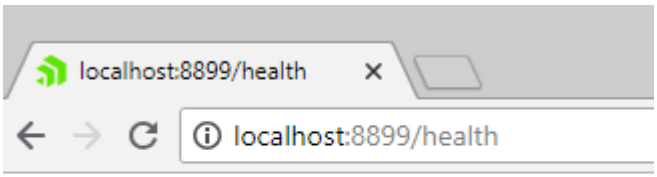
Return the HealthScanner status code

Typically used with the ELB health check, http://*hostname*:8899/health can be used to return a status code for the server.

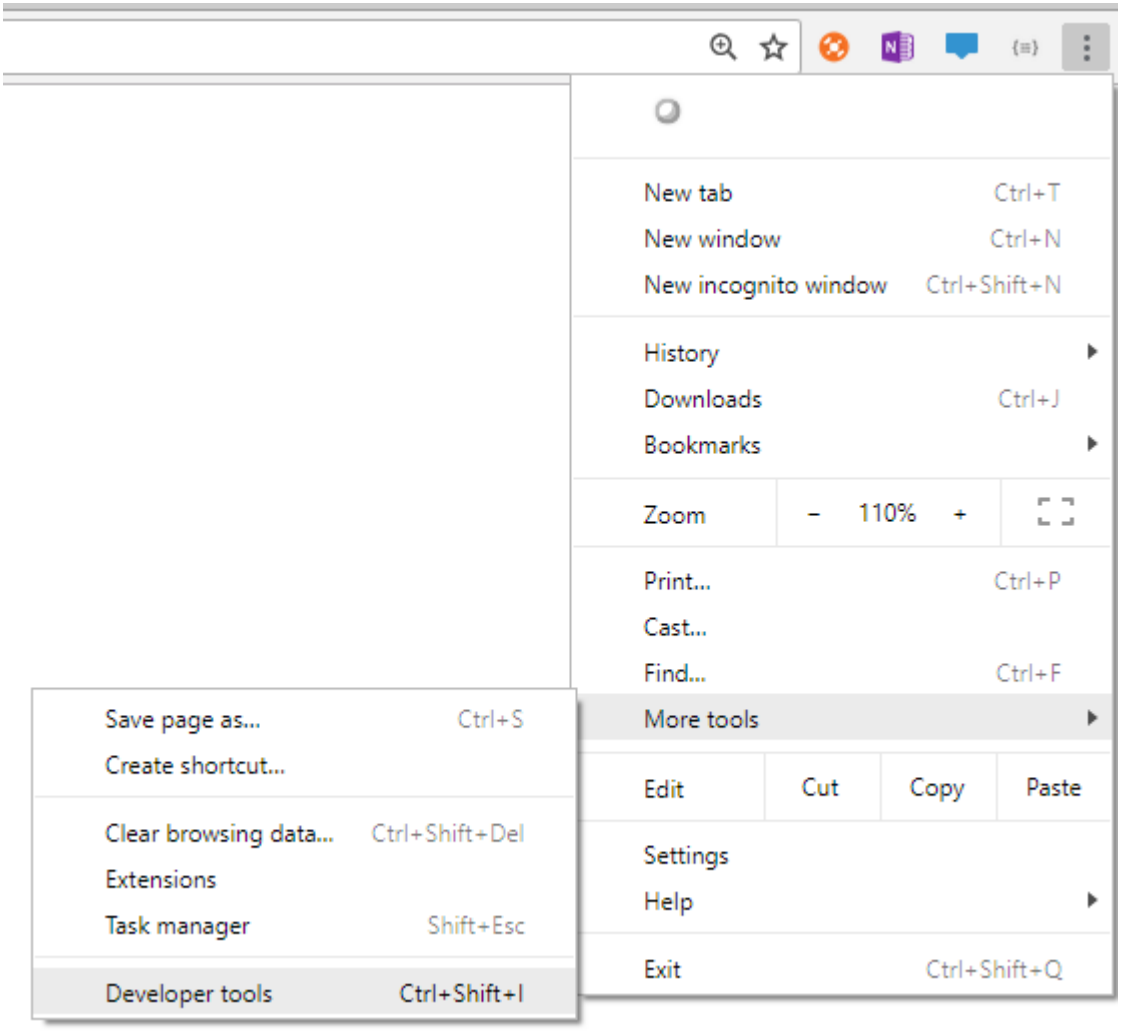
1. Enter the following URL in a browser:

   ```
   http://localhost:8899/health
   ```

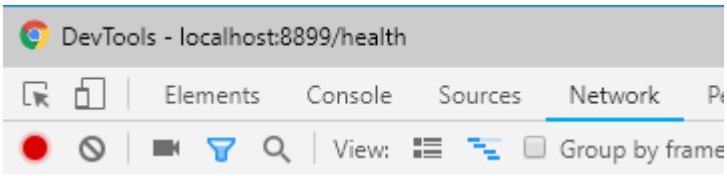
2. A success code of 200 or a blank response is returned depending on the browser shown:



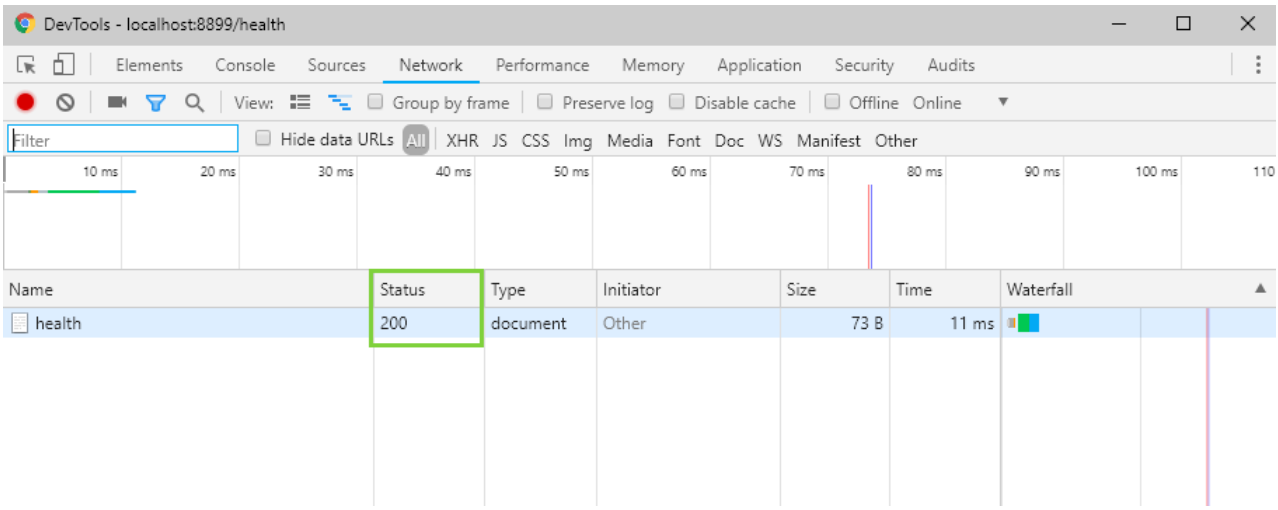
3. To show the status code in Google Chrome, select **…, More Tools, Developer tools**:



4. Select **Network**:



5. Refresh the URL.
6. Return to the DevTools browser window. Note that a **Status** of **200** indicates that the HealthScanner is available.



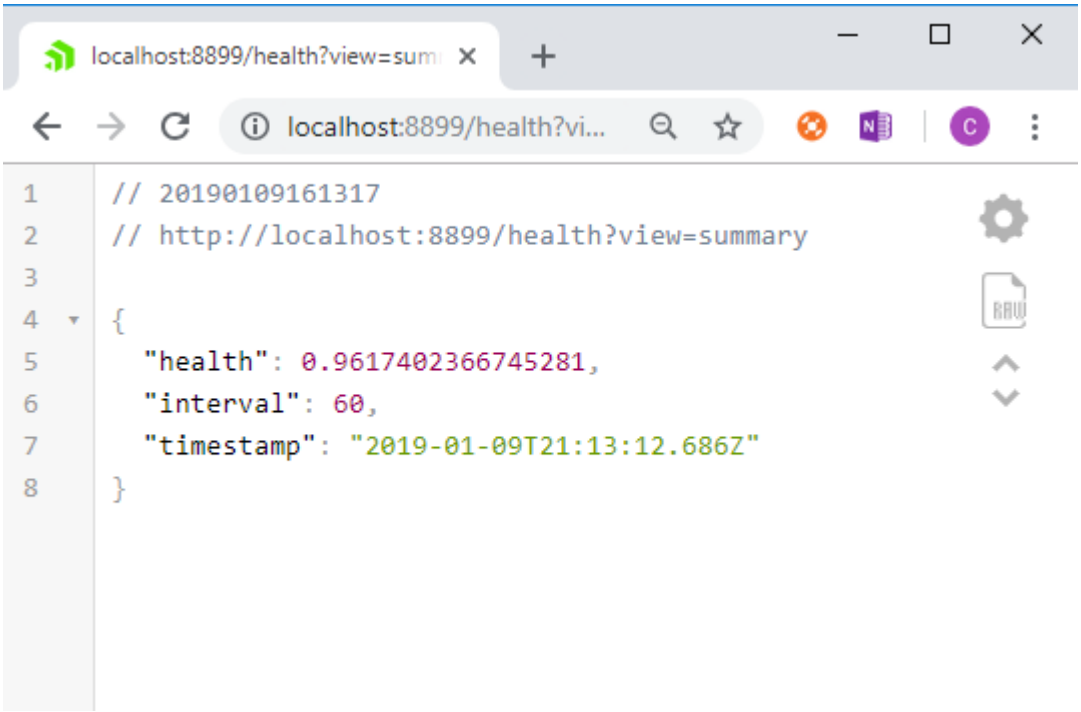
Return JSON summary view

To return a JSON summary of the overall health score:

1. Enter the following URL:

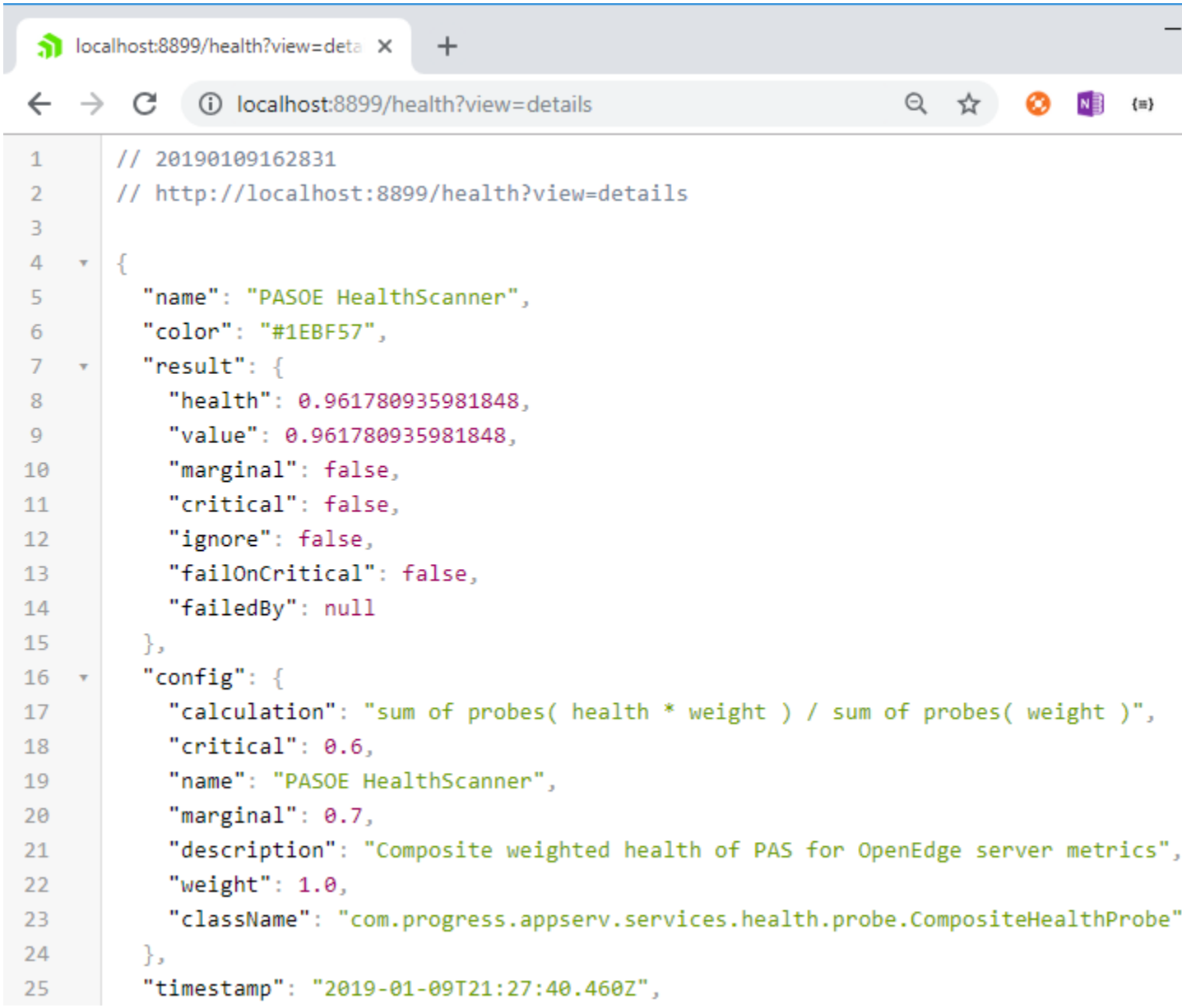
```
http://hostname:8899/health?view=summary
```

2. Locate the `health` value that represents the overall health of the server. In this example, **.96** is the equivalent of **96%**, or good health, based on the probes.



Return JSON details view

For a complete JSON representation of all the probes, use http://hostname:8899/health?view=details, which returns all the probes including details. The overall health is still **96%**. This request shows how the overall health was calculated for the summary value.



Return as an OEJMX query

You can monitor and manage instances through scripts using the OpenEdge JMX (OEJMX) command-line utility to connect to a local PAS for OpenEdge instance. Based on Java Management Extensions (JMX) OEJMX connects to an instance using Java objects called Managed Beans (MBeans). One or more MBeans instrument a resource using attributes and operations. Direct access to the underlying MBeans provides you with the freedom to write scripts to monitor and manage applications in ways that may not be available using OpenEdge Management or JConsole.

For more information about OEJMX, see Use OEJMX to manage and monitor an instance.

1. Use a `proenv` command prompt to change to the instance directory:

```
proenv>cd instance-name
```

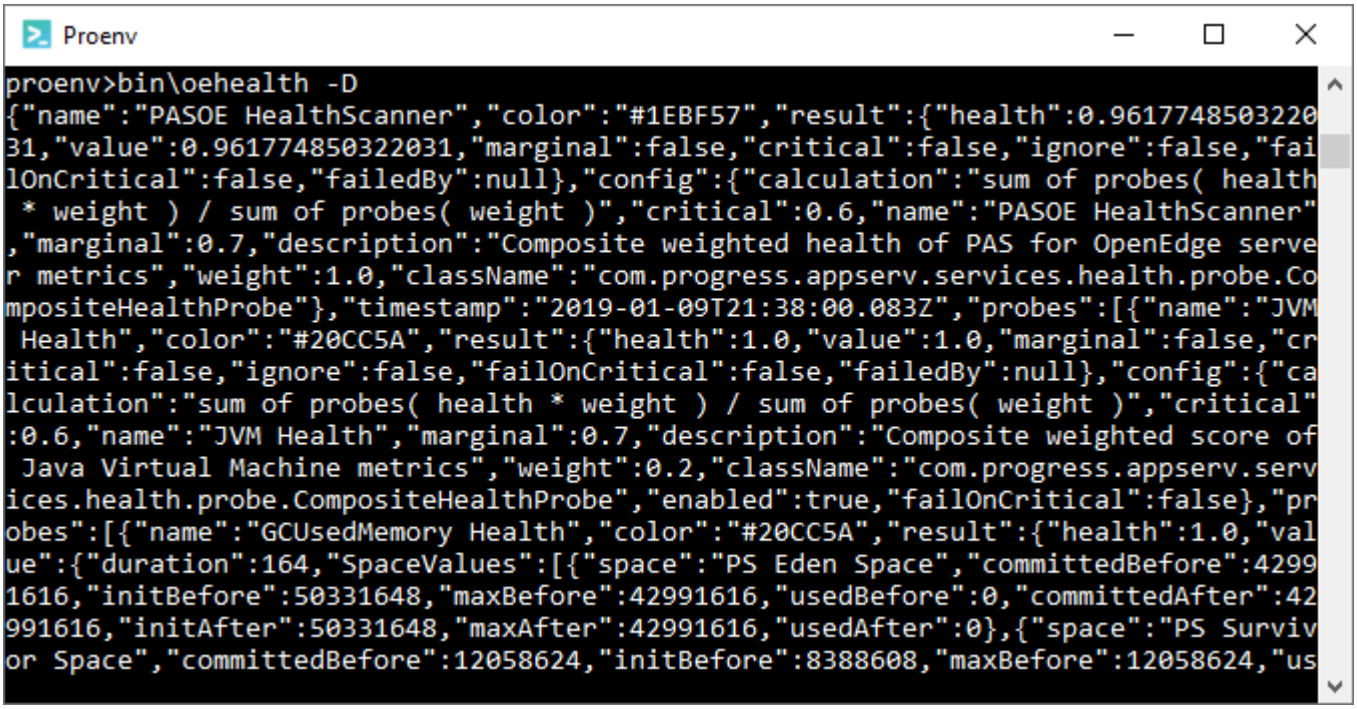
2. Request the overall health score:

```
proenv>bin\oehealth
```



3. Display the detailed JSON for the server health:

```
proenv>bin\oehealth -D
```



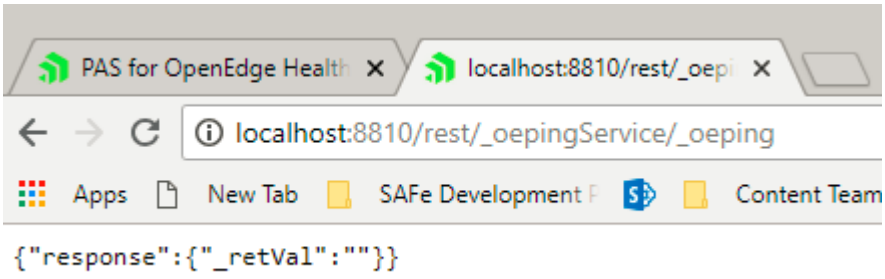
Note: These commands can be included in scripts to monitor your server.

Display a healthy score for a REST call

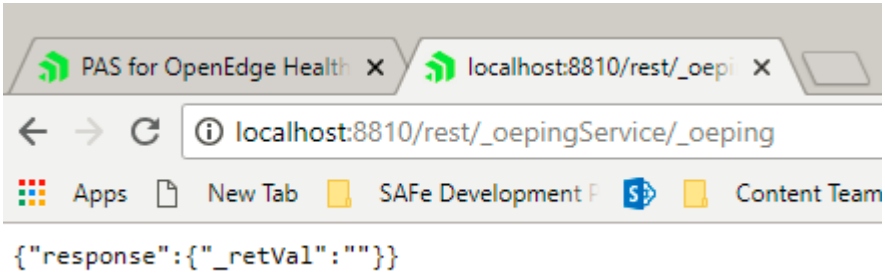
The probes collect data after they are activated. To activate the probes for a REST call, you can use the `_oeping` service that is shipped with the product. This service activates the REST probes. In a later example, a request will be made to an unhealthy server and different results will be displayed.

1. In a separate tab of the browser, make a ping service REST call using the following URL:

```
localhost:8810/rest/_oepingService/_oeping
```



2. Confirm that the `"_retVal":""` was returned.

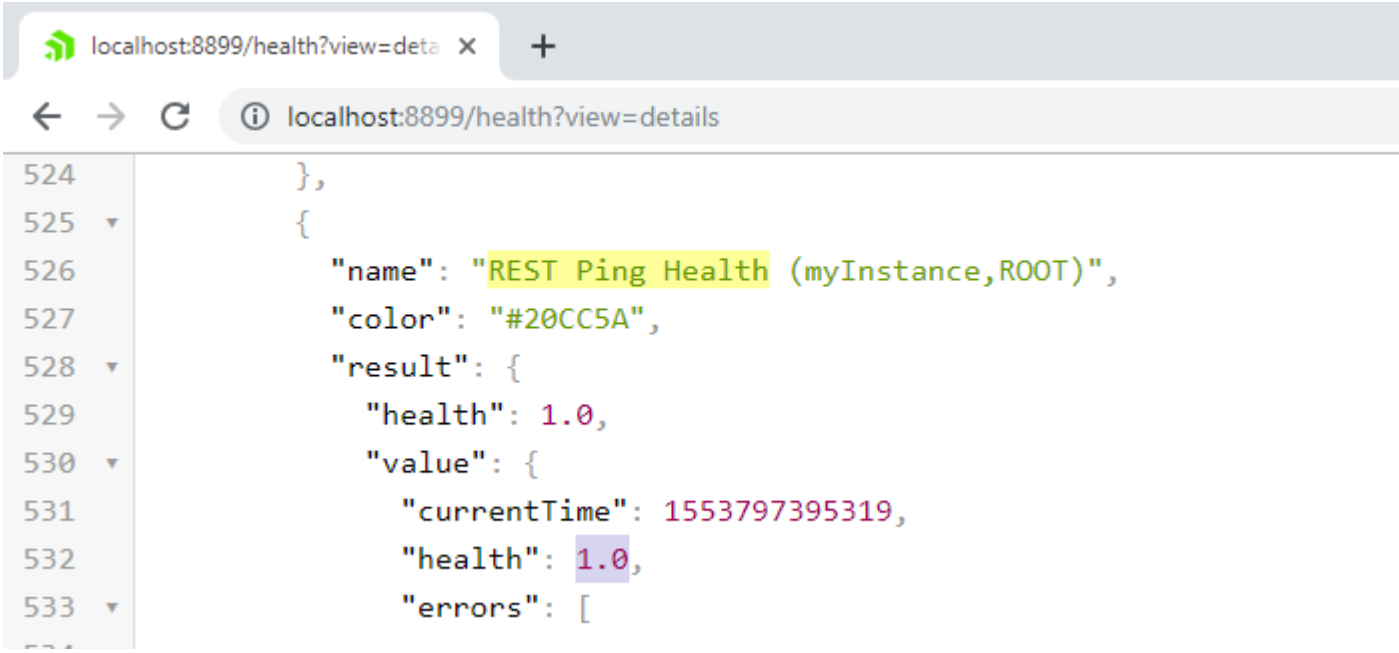


Note:

For JSON formatted return values, add the appropriate JSON formatting plugin for your browser.

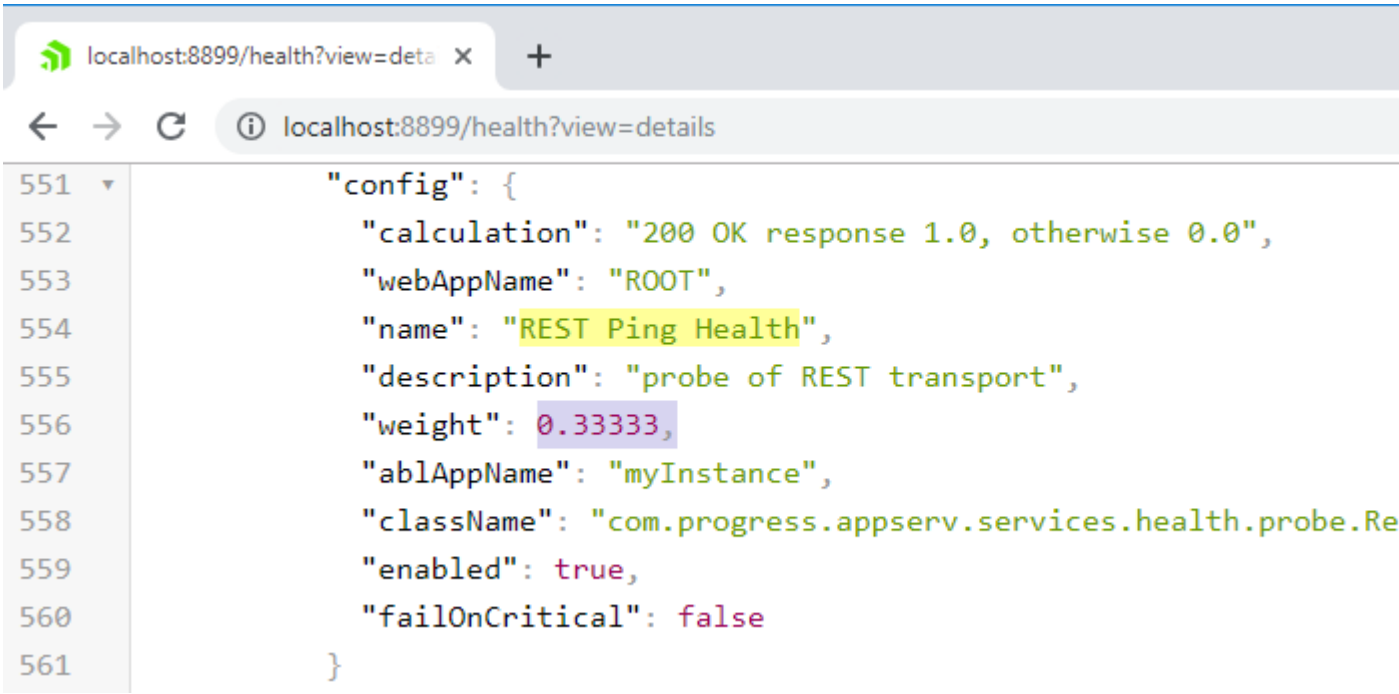


3. Poll the details of the the probes again with http://hostname:8899/health?view=details .
4. Press **Ctrl+F** to search for **REST Ping Health**. Stop at the first occurrence.



The value of `1` for the `health` is **100%** healthy for the **REST Ping Health** .

5. In the second occurrence, see how the probe is weighted.



The **weight** this has on the score is **0.33333**.

The value of `1` for the `health` is **100%** healthy for the **REST Ping Health** .

Display an unhealthy score for a REST call

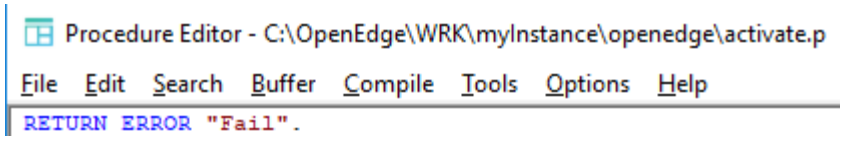
In this topic, an error is introduced to see how it affects the health score of the server. Edit the `openedge.properties` file, and add an `activate.p` procedure as the **session activate** procedure. This procedure is run when a session is started by REST call. In this example, the procedure explicitly raises an error so the HealthScanner will detect the problem.

1. Use an editor to modify the `openedge.properties` file located in the *`instance-name`*`\conf` directory to specify `sessionActivateProc=activate.p`.

```
[AppServer.Agent]
    agentMaxPort=62202
    agentMinPort=62002
    agentShutdownProc=
    agentStartupProc=
    agentStartupProcParam=
    binaryUploadMaxSize=0
    collectStatsData=0
    fileUploadDirectory=
    flushStatsData=0
    infoVersion=9010
    keyAlias=
    keyAliasPasswd=
    # keyStorePasswd=
    keyStorePath=${DLC}/keys/
    lockAllExtLib=
    lockAllNonThreadSafeExtLib=
    noSessionCache=0
    numInitialSessions=5
    PROPATH=${CATALINA_BASE}/openedge,${DLC}/tty,${DLC}/tty/netlib/OpenEdge.Net.pl
    sessionActivateProc=activate.p
    sessionConnectProc=
    sessionDeactivateProc=
    sessionDisconnProc=
    sessionExecutionTimeLimit=0
    sessionShutdownProc=
    sessionStartupProc=
    sessionStartupProcParam=
    sessionTimeout=180
    sslAlgorithms=
    usingThreadSafeExtLib=
    uuid=
    workDir=${CATALINA_BASE}/work
```

2. Save changes to the `openedge.properties` file.
3. Use the editor to create an `activate.p` procedure that returns an error and a string:

```
RETURN ERROR "Fail".
```

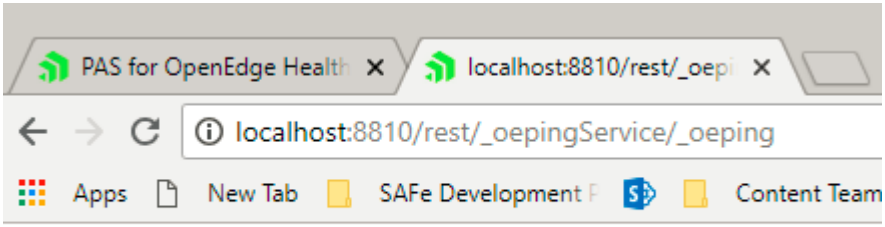


4. Save the file to *`instance`*`\openedge\activate.p`, which puts the file in the PROPATH for the instance.

5. Restart the server:

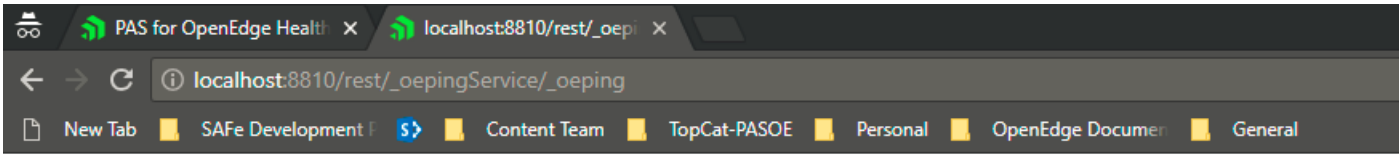
```
tcman pasoestart -restart
```

6. Test the REST service again.




```
{"response":{"_retVal":""}}
```

7. This time there is an error introduced by the `activate.p` procedure. The unformatted output:




```
{"_errors":[{"_errorMsg":"ERROR condition: The Server application has returned an error. (7243) (7211)","_errorNum":0}],"
```

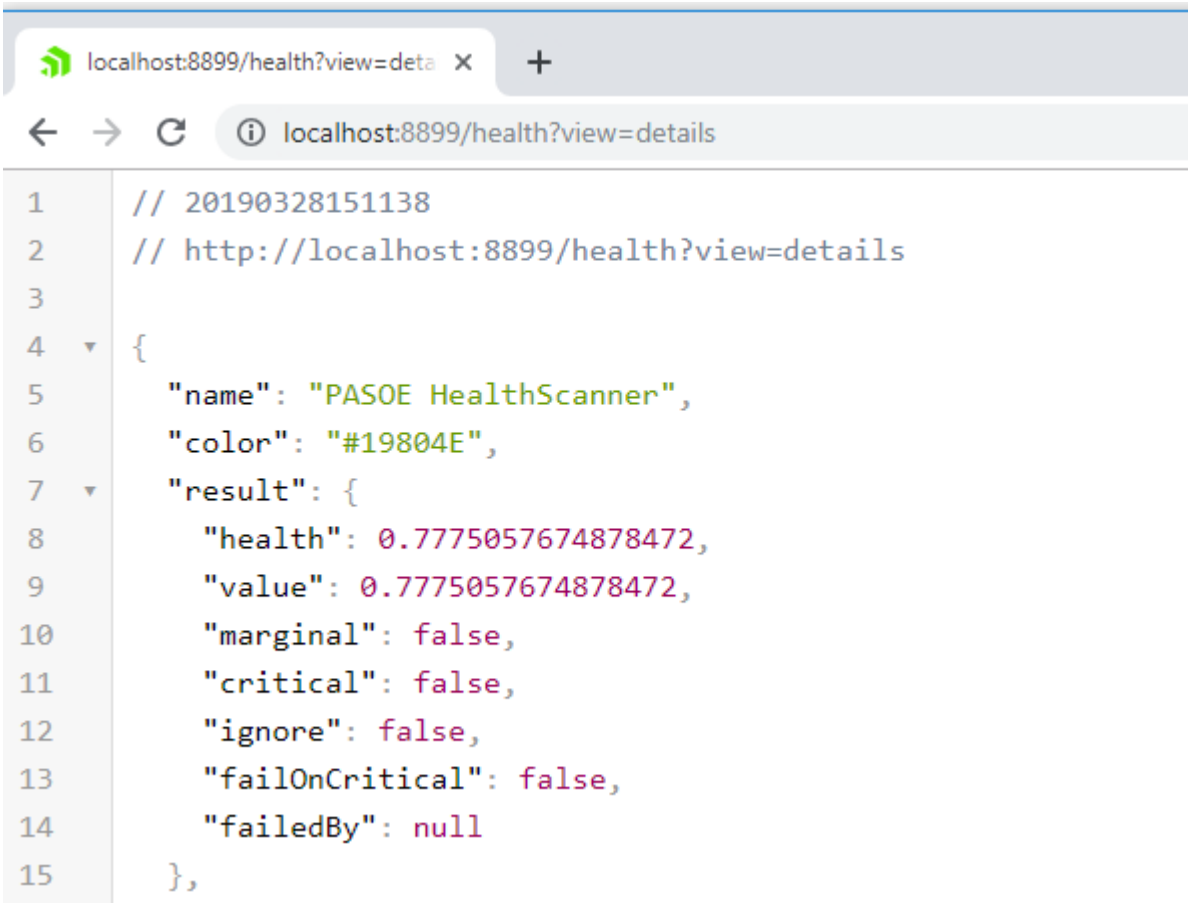

The formatted output:




```
// 20180731085707
// http://localhost:8810/rest/_oepingService/_oeping

{
  "_errors": [
    {
      "_errorMsg": "ERROR condition: The Server application has returned an error. (7243) (7211)",
      "_errorNum": 0
    }
  ],
  "_retVal": "Fail"
}
```


8. Poll the details of the probes again with http://hostname:8899/health?view=details .



As a result, the **PASOE HealthScanner** score lowered to **77%**.

9. Press **Ctrl+F**to search for **REST Ping Health**. Stop at the first occurrence.



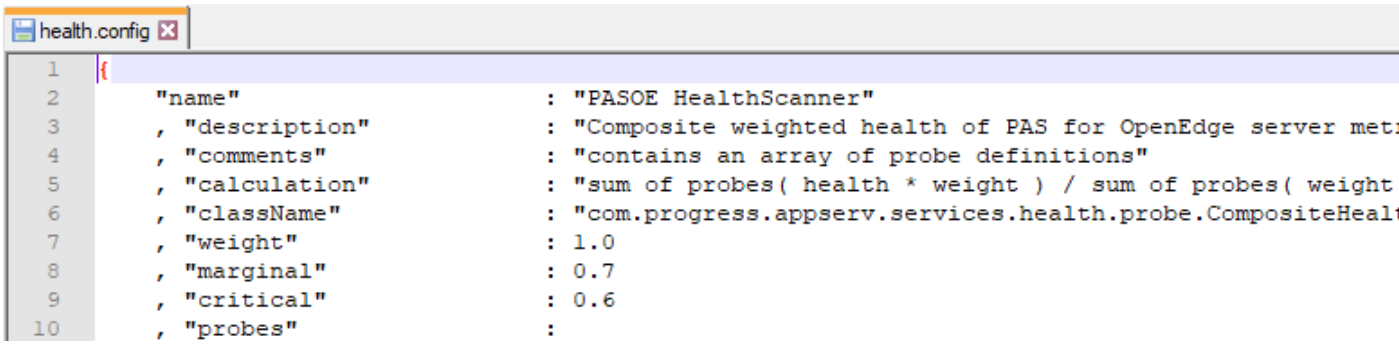
Note: The value of `0` for the `health` is **0%** healthy for the **REST Ping Health** . The HealthScanner acts as the warning light not a diagnostic tool for analyzing the issue. The next steps is to review the log files to discover the underlying cause of the lowered health score.

10. Modify the `openedge.properties` file located in the *`instance-name`*`\conf` directory to reset the value for `sessionActivateProc=` to the original blank value to avoid any future problems.

Customize HealthScanner reports

Use the *`instance-name`*`\conf\health.config` file, a JSON formatted file, to customize what the HealthScanner reports for each instance. There are settings for the overall health score and thresholds at the top. Individual probes have unique values in the subsequent entries.

1. Open the *`instance-name`*`\conf\health.config` file in Notepad++ or another JSON formatting text editor.



Note: The overall weight, marginal, and critical values appear at the top for Whole Server Health. Values above marginal are green. Values between marginal and critical are yellow, and values below critical are red.

2. Locate the **Transports Health** details, in particular the **REST Ping Health** details. The weight is **.33333** of the overall Transports Health score.

```
{
    "name"                  : "REST Ping Health"
    , "description"         : "probe of REST transport"
    , "comments"            : ""
    , "calculation"         : "200 OK response 1.0, otherwise 0.0"
    , "className"           : "com.progress.appserv.services.health.probe.RestTransp
    , "weight"              : 0.33333
    , "failOnCritical"      : false
    , "enabled"             : true
}
```

Note: If you are only using one transport type on a server, you may opt to change these settings. For example, if only two transports are used, then those transports would each be a weight of .50 and the third transport would be 0. If the server will support services on all three transports it should be kept the same.

3. Close the file without saving the changes.
4. To learn more about the available settings and specific probes, review the *instance-name*\conf\health.config.README file.

```
# -------------------- health.config.README -----------------------------
#
# README for the configuration file for the PASOE HealthScanner
#
##############################################################################
##
------------------------------------------------------------------------------
The health.config file is JSON and as such, must be valid JSON. Please
verify all changes with a JSON validator.
```

Write a custom HealthScanner view

A pluggable interface is required for the OpenEdge HealthScanner to output data to other monitoring tools. You can write a custom HealthScanner view to output HealthScanner details in the format of your choice.

The plugin architecture in PAS for OpenEdge is based on the Java Service Provider Interface (SPI). PAS for OpenEdge requires implementing the correct interfaces for formatting the HealthScanner data and packaging the implementation in a JAR file.

The OpenEdge HealthScanner has four built-in views that can be accessed by URLs in the following form:

- `http://localhost:`*`port`*`/health?view=details`
- `http://localhost:`*`port`*`/health?view=summary`
- `http://localhost:`*`port`*`/health?view=config`
- `http://localhost:`*`port`*`/health?view=dataset`

This topic describes how to create additional views when a specific format is needed for the HealthScanner data. For example, a new view called Acme might be called using the `view=acme` parameter in the URL.

To write a custom HealthScanner view that can output data to other monitoring tools, perform the following steps:

1. Implement the `com.progress.appserv.services.health.spi.HealthViewVisitor` interface. This interface follows a hierarchical visitor pattern and is available in `oecollector-`*`version`*`.jar`:

```
public interface HealthViewVisitor
{
 // Called for each probe that is not a composite.
 void visit ( HealthProbe probe );

 // Called for each composite on entering.
 void enter ( CompositeHealthProbe probe );

 // Called for each composite on leaving.
 void leave ( CompositeHealthProbe probe );

 // Method to get result
 String getDetails ();
}
```

2. Implement the `com.progress.appserv.services.health.spi.HealthViewFactory` interface. This is a factory that creates instances of the view visitor interface that you implemented in Step 1. The `HealthViewFactory` interface is also included in `oecollector-`*`version`*`.jar`:

```
public interface HealthViewFactory
{
 // Provide the name of the view visitor.
 String getViewName ();

 // Get an instance of the view visitor.
 HealthViewVisitor getVisitor ();
}
```

3. Register the service provider by creating a file named `com.progress.appserv.services.health.spi.HealthViewFactory`. The content of the file is the name of your factory class.

   For example, if your view factory implementation from Step 2 contains the line `com.acme.AcmeViewFactory`, then your factory class is `com.acme.AcmeViewFactory`.

4. Package the service as a JAR file. Include compiled versions of the classes that you created in Step 1 and Step 2. As well as the file you created in Step 3.

   For example, if your visitor interface is `com.acme.AcmeViewVisitor` and your factory is `com.acme.AcmeViewFactory` then the contents of the JAR file looks like this:

```
META-INF/MANIFEST.MF
META-INF/services/com.progress.appserv.services.health.spi.HealthViewFactory
com/acme/AcmeViewFactory.class
com/acme/AcmeViewVisitor.class
```

5. Copy the JAR file into `$DLC/servers/pasoe/lib/ext`.
6. Restart the PAS for OpenEdge instance, and test the view in a browser:

```
http://localhost:port/health?view=acme
```

Example interface implementation

The following code is an example of a HealthScanner visitor interface implementation called **AcmeViewVisitor.java**:

```
package com.acme;

import java.util.ArrayDeque;
import java.util.Deque;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ArrayNode;
import com.fasterxml.jackson.databind.node.ObjectNode;
import com.progress.appserv.services.health.probe.CompositeHealthProbe;
import com.progress.appserv.services.health.probe.HealthProbe;
import com.progress.appserv.services.health.spi.HealthViewVisitor;
import com.progress.appserv.services.health.probe.ProbeResult;

/**
 * A probe visitor that creates a JSON document from the polling results
 * of a tree of probes.
 */
public class AcmeViewVisitor
    implements HealthViewVisitor
{
    private ObjectNode json = null;
    private Deque<ArrayNode> ancestors;
    private ArrayNode current = null;

    // Constants that are specific to the JSON created by this visitor
    private static final String NAME = "name";
    private static final String HEALTH = "health";
    private static final String CHILDREN = "children";

    // JSON mapper.
    private ObjectMapper mapper;

    /**
     * Constructor.
     */
    public AcmeViewVisitor()
    {
        // Create a mapper for creating JSON nodes.
        this.mapper = new ObjectMapper();

        // Create a stack for keeping track of ancestor layers.
        this.ancestors = new ArrayDeque<>();
    }

    /**
     * Get a JSON description of the detailed results of polling the probe tree
     *
     * @return  A JSON document.
     */
    @Override
    public String getDetails()
    {
        // Guard for visitor not yet accepted.
        if (this.json == null)
            return null;

        String retval = null;

        try
        {
            retval = this.mapper.writeValueAsString( this.json );
        }
        catch (JsonProcessingException e)
```

```
        {
            retval = "{\"message\": \"Details unavailable.\"}";
        }

        return retval;
    }

    /**
     * Called for each probe that is not a composite.
     *
     * @param  The probe to visit.
     */
    @Override
    public void visit( HealthProbe probe )
    {
        createJsonNode( probe );
    }

    /**
     * Called for each composite on entering
     *
     * @param probe  The composite probe.
     */
    @Override
    public void enter( CompositeHealthProbe probe )
    {
        ObjectNode probeNode = createJsonNode( probe );
        ArrayNode children = mapper.createArrayNode();
        probeNode.set( CHILDREN, children );

        if (this.current != null)
            this.ancestors.push( this.current );
        this.current = children;
    }

    /**
     * Called for each composite on leaving, after traversing the child probes.
     *
     * @param  the composite probe
     */
    @Override
    public void leave( CompositeHealthProbe probe )
    {
        this.current = this.ancestors.isEmpty() ? null : this.ancestors.pop();
    }

    /**
     * Create a json object node from a probe polling result.
     *
     * @param probe  The probe to get result from
     * @return  A JSON object node.
     */
    private ObjectNode createJsonNode( HealthProbe probe )
    {
        // Create the main node with name
        ObjectNode probeNode = mapper.createObjectNode();
        probeNode.put( NAME, probe.getName() );

        ProbeResult result = probe.getPollResult();
        if (result != null)
            probeNode.put( HEALTH, result.health() );

        // Add to the parent
        addJsonNodeToParent( probeNode );

        return probeNode;
    }

    /**
```

```
     * Add the given node to the current list of children.
     *
     * @param probeNode  The node to add.
     */
    private void addJsonNodeToParent( ObjectNode probeNode )
    {
        if (current == null)
            // Must be the root. Save it as the result of this visitor
            this.json = probeNode;
        else
            // Not the root.  Add to array of children of current composite.
            this.current.add( probeNode );
    }
}
```

The following code is an example of a HealthScanner factory interface implementation called **AcmeViewFactory.java**:

```
package com.acme;

import com.progress.appserv.services.health.spi.HealthViewFactory;
import com.progress.appserv.services.health.spi.HealthViewVisitor;

public class AcmeViewFactory
    implements HealthViewFactory
{
    private final static String NAME = "acme";

    public String getViewName()
    {
        return NAME;
    }

    public HealthViewVisitor getVisitor()
    {
        return new AcmeViewVisitor();
    }
}
```

Note: The HealthScanner service provider interfaces are available in the `oecollector-`*`version`*`.jar` that is included with your PAS for OpenEdge distribution in the `$DLC/servers/pasoe/lib/ext` directory.

Collect runtime metrics

PAS for OpenEdge allows for the collection of runtime metrics for the Session Manager and the ABL web application through the `collectMetrics` property in the `openedge.properties` file.

Metrics can be collected in the form of counts and times that are gathered over a period of time. Anything that is valid only for a snapshot of time is not a metric. Examples of metrics can be, for example:

- Number of hits to the web application
- Average response time
- Number of fatal errors

Collecting metrics, especially time-based metrics, can be expensive for system resources. The `openedge.properties` file lets administrators enable or disable the collection of time-based and count-based metrics.

This article describes how to configure the collect metrics functionality in PAS for OpenEdge for both the Session Manager and ABL web applications.

Note: All the time metrics are collected in milliseconds.

Enable collect metrics

Both the Session Manager and ABL web applications enable collecting metrics using the `openedge.properties` file. Both properties support the following values for collecting metrics:

```
collectMetrics=
 0 - disable the collection of metrics.
 1 - enable the collection of count-based metrics.
 2 - enable the collection of time-based metrics.
 3 - enable the collection of both count-based and time-based metrics.
```

Configure Session Manager runtime metrics

Session Manager runtime metrics represent the metrics of the ABL application. Each PAS for OpenEdge instance contains a default ABL application in the `/ablapps` directory with the same name as the instance (for example, `oepas1`).

To configure the Session Manager to collect runtime metrics for the ABL application, perform the following steps:

1. Open the `openedge.properties` file for the target PAS for OpenEdge instance.
2. Under the `[AppServer]` section, change the value of `collectMetrics` to the appropriate value (`1`, `2`, or `3`).
3. Restart the PAS for OpenEdge instance.
4. View the Session Manager metrics using the oemanager REST API or OEJMX query.

   For more information, see the following topics:

   - "Get runtime metrics" REST API (requires oemanager to be deployed to the instance)
   - OEJMX "Get metrics for an ABL application query"

   You can also reset the runtime metrics using the "Reset runtime metrics" REST API or OEJMX "Reset metrics for an ABL application query".

Session Manager metrics

Setting `collectMetrics` to `1` returns the following (count-based) metrics:

Session Manager count metric	Description
`requests`	Total number of requests to the session.
`writeErrors`	Number of errors that occurred during writing a request to the agent.
`reads`	Number of times a response was read by the session from the agent.
`concurrentConnectedClients`	Number of clients connected at a particular time.
`writes`	Number of times requests were written by the session on the agent.
`maxConcurrentClients`	Maximum number of concurrent clients.
`readErrors`	Number of errors that occurred while reading a response from the agent.
`numReserveABLSessionTimeouts`	Number of timeouts that occurred while reserving a local ABL session.
`numReserveABLSessionWaits`	Number of waits that occurred while reserving a local ABL session.

Setting `collectMetrics` to `2` returns the following (time-based) metrics:

Session Manager time metric	Description
`maxAgentReadTime`	Maximum time to read a response from the agent.
`avgReserveABLSessionWaitTime`	Average time that a reserved ABL session had to wait before executing.
`totReserveABLSessionWaitTime`	Total time that reserved ABL sessions had to wait before executing.
`stdDevAgentReadTime`	Standard deviation of the time taken to read a response from the agent.
`maxReserveABLSessionWaitTime`	Maximum time that a reserved ABL session had to wait before executing.
`maxConcurrentClients`	Maximum number of concurrent clients.
`avgAgentReadTime`	Average time to read a response from the agent.
`minAgentReadTime`	Minimum time to read a response from the agent.

Setting `collectMetrics` to 3 returns both the count-based and time-based metrics.

Configure ABL web application runtime metrics

ABL web application runtime metrics represent the metrics of the web application and its supporting transports. The default ABL web application for the PAS for OpenEdge instance is ROOT. ABL web applications are located in the `/webapps` directory of the instance.

The metrics are obtained for each enabled transport on the ABL web application.

To configure an ABL web application to collect runtime metrics, perform the following steps:

1. Open the `openedge.properties` file for the target PAS for OpenEdge instance.
2. Under the `[ABL_app_name.Web_app_name]` section, change the value of `collectMetrics` to the appropriate value.

   For example, to enable collect metrics in the default instance's ABL application (oepas1) for the ROOT application, navigate to the `[oepas1.ROOT]` section of the `openedge.properties` file.
3. Restart the PAS for OpenEdge instance.
4. Get the ABL web application metrics for each transport. For example, if you have the oemanager application deployed to your PAS for OpenEdge instance, the metrics for each transport are available at the following API endpoints:

   ```
   http://localhost:port/oemanager/applications/oepas1/webapps/ROOT/transports/apsv/metrics
   http://localhost:port/oemanager/applications/oepas1/webapps/ROOT/transports/rest/metrics
   http://localhost:port/oemanager/applications/oepas1/webapps/ROOT/transports/soap/metrics
   http://localhost:port/oemanager/applications/oepas1/webapps/ROOT/transports/web/metrics
   ```

   For more information, see the following links:

   - REST APIs (require oemanager to be deployed to the instance):
     - Get runtime metrics of a transport
     - Reset runtime metrics of a transport
   - OEJMX Queries:
     - Get transport metrics for an ABL web application query
     - Reset transport metrics for an ABL web application query

APSV transport metrics

Setting `collectMetrics` to `1` for the ABL web application returns the following (count-based) metrics for the APSV transport:

APSV count metric	Description
`statusRequests`	Number of requests for the status of APSV transport.
`connectRequests`	Number of requests for new connections to the session.
`sessionErrors`	Number of errors getting a session to run a remote procedure.
`connectErrors`	Number of connect errors.
`disconnectRequests`	Number of disconnect requests.
`sessionRequests`	Number of requests to get a local session to run a remote procedure.
`disconnectErrors`	Number of errors during disconnect.
`forbiddenErrors`	Number of errors due to transport being disabled.

Setting `collectMetrics` to `2` does not return time-based metrics.

Setting `collectMetrics` to `3` returns the count-based metrics.

SOAP transport metrics

Setting `collectMetrics` to `1` for the ABL web application returns the following (count-based) metrics for the SOAP transport:

SOAP count metric	Description
`statusRequests`	Number of requests for the status of the SOAP transport.
`serviceDisabled`	Number of WSDL or SOAP requests that were denied because the WSDL or Web Service Application services were disabled by the administrator.
`soapProcessorErrors`	Number of administration and SOAP requests that failed due to an internal SOAP processor related error.
`httpRequestErrors`	Number of malformed HTTP requests.
`httpRequests`	Number of HTTP requests that are not WSDL or SOAP related.
`methodNotAllowederrors`	Number of HTTP methods that are not supported by the WSA. The WSA accepts only GET and POST HTTP methods.
`soapRequests`	Number of incoming SOAP requests.
`wsdlRequests`	Number of requests to fetch the WSDL document.
`activeRequests`	Number of in-process administration or Web Service Application SOAP requests.
`urlNotFoundErrors`	Number of HTTP requests that requested a URL path not supported by the SOAP transport.

Setting `collectMetrics` to 2 returns the following (time-based) metrics:

SOAP time metric	Description
`minTotalTime`	Minimum transport time for SOAP requests to successfully return to the client via the transport after making a round trip to PAS for OpenEdge.
`stdDevTotalTime`	Standard deviation in the transport time for SOAP requests to successfully return to the client via the transport after making a round trip to PAS for OpenEdge.
`avgTotalTime`	Average transport time for SOAP requests to successfully return to the client via the transport after making a round trip to PAS for OpenEdge.
`maxTotalTime`	Maximum transport time for SOAP requests to successfully return to the client via the transport after making a round trip to PAS for OpenEdge.

Setting `collectMetrics` to 3 returns both the count-based and time-based metrics for the SOAP transport.

REST transport metrics

Setting `collectMetrics` to `1` for the ABL web application returns the following (count-based) metrics for the REST transport:

REST count metric	Description
`statusRequests`	Number of requests for the of status of the REST Transport (i.e. requests at `/rest` URL).
`successfulRunRequests`	Number of run requests that successfully returned from PAS for OpenEdge to the REST transport.
`requests`	Total number of requests coming to the REST transport including status requests.
`serviceUnavailableRequests`	Number of requests that came when the REST Transport was disabled (i.e. when the service was unavailable).
`failedRequests`	Number of requests that returned with a non-success return code to the REST client.
`connectRequests`	Number of requests for which new connections were made between the REST transport's endpoint and PAS for OpenEdge. **Note:** The REST transport normally reuses the same connection object, so you may find fewer connect requests than connections.
`expressionErrors`	Number of requests for which the parameter value had ill-formatted JSON data.
`successfulConnectRequests`	Number of requests for which connection objects were created successfully. These successful connection objects are used to run the actual service on PAS for OpenEdge.
`successfulRequests`	Number of requests that made a successful round-trip to PAS for OpenEdge and returned successfully with a 200 OK response code.
`runRequests`	Number of requests for which an ABL procedure, class, or function were invoked on PAS for OpenEdge.
`serviceNotFoundErrors`	Number of requests for which no REST service was found.

Setting `collectMetrics` to 2 returns the following (time-based) metrics:

REST time metric	Description
`avgConnectTime`	Average time to establish a connection to PAS for OpenEdge from the REST Transport.
`minDisconnectTime`	Minimum time it took for a disconnect request to run.
`stdDevDisconnectTime`	Standard deviation of the time it took for a disconnect request to run.
`stdDevConnectTime`	Standard deviation of the time it takes to establish a connection to PAS for OpenEdge from the REST Transport.
`stdDevSessionTime`	Standard deviation of the time it took for a session to run a remote procedure.
`maxConnectTime`	Maximum time it took to establish a connection to PAS for OpenEdge from the REST Transport.
`minSessionTime`	Minimum time it took for a session to run a remote procedure.
`avgSessionTime`	Average time it took for a session to run a remote procedure.
`maxDisconnectTime`	Maximum time it took for a disconnect request to run.
`minConnectTime`	Minimum time it took to establish a connection to PAS for OpenEdge from the REST Transport.
`avgDisconnectTime`	Average time it took for a disconnect request to run.
`maxSessionTime`	Maximum time it took for a session to run a remote procedure.

Setting `collectMetrics` to 3 returns both the count-based and time-based metrics.

WEB transport metrics

Setting `collectMetrics` to `1` for the ABL web application returns the following (count-based) metrics for the WEB transport:

WEB count metric	Description
`patchErrors`	Number of PATCH request errors.
`postErrors`	Number of POST request errors.
`failedServletRequests`	Number of failed requests to the WEB transport.
`getRequests`	Number of GET requests.
`optionsErrors`	Number of OPTIONS request errors.
`connectRequests`	Indicates if the first WEB request was received. This metric is either 0 or 1.
`putRequests`	Number of PUT requests.
`getErrors`	Number of GET request errors.
`deleteRequests`	Number of DELETE requests.
`patchRequests`	Number of PATCH requests.
`ablConnectErrors`	Number of failures to create the `OpenAppObject` for transport communication with the ABL.
`postRequests`	Number of POST requests.
`successfulServletRequests`	Number of successful requests to the WEB transport.
`optionsRequests`	Number of OPTIONS requests.
`traceRequests`	Number of TRACE requests.
`putErrors`	Number of PUT requests.
`deleteErrors`	Number of DELETE request errors.
`headRequests`	Number of HEAD requests.
`traceErrors`	Number of TRACE request errors.
`servletRequests`	Total number of requests to the WEB transport.
`headErrors`	Number of HEAD request errors.

Setting `collectMetrics` to 2 returns the following (time-based) metrics:

WEB time metric	Description
`maxABLConnectTime`	Maximum time it took to create an ABL connection using the `OpenAppObject`.
`stdABLDevConnectTime`	Standard deviation of the time it took to create an ABL connection using the `OpenAppObject`.
`minABLProcessingTime`	Minimum amount of time it took the WEB transport to process a request.
`maxABLProcessingTime`	Maximum amount of time it took the WEB transport to process a request.
`stdDevABLProcessingTime`	Standard deviation of the amount of time it took the WEB transport to process a request.
`minABLConnectTime`	Minimum time it took to create an ABL connection using the `OpenAppObject`.
`avgABLProcessingTime`	Average time it took the WEB transport to process a request.
`avgABLConnectTime`	Average time it took to create an ABL connection using the `OpenAppObject`.

Setting `collectMetrics` to 3 returns both the count-based and time-based metrics.

Enable debugging

Debugging support for PAS for OpenEdge environments is provided by the `oedbg.war` application. You can only debug code running on PAS for OpenEdge if it is in a development environment.

System administrators can enable debugging for development instances of PAS for OpenEdge by using the proDebugEnable utility.

To run the proDebugEnable command-line utility:

1. Run the Proenv command shell.
2. Enable debugging:

   ```
   prodebugenable -enable-all
   ```

   A confirmation message indicates that debugging is enabled. You set the `ENABLE_OPENEDGE_DEBUGGER` environment variable at the operating system level. When you do so, set the value of the variable as either yes or true.

Debugging remains enabled unless you explicitly disable it. If you did not set the ENABLE_OPENEDGE_DEBUGGER environment variable, then you can disable debugging simply by entering the following command from the Proenv window:

```
prodebugenable -disable-all
```

A confirmation message indicates that debugging is disabled. If you set the ENABLE_OPENEDGE_DEBUGGER environment variable, you must also remove the variable.

To enable debugging for a PAS for OpenEdge development instance:

1. Shut down PAS for OpenEdge instance if it is running. In order for the debugger to be effective, the instance must be started using the following instructions.
2. Open a program that you want to debug in Progress Developer Studio and set a breakpoint in it by double-clicking along the left-edge of the line where you want to start debugging. A dot (.) appears to the left of the line if you've successfully set a breakpoint.
3. From the Developer Studio menu, select **Run** > **Debug Configurations**.
4. Expand the Progress Application Server for OpenEdge node in the left pane and select your instance.
5. Set the appropriate properties on the property sheet and click **Debug** to start the instance.

   See "Debug Configuration Settings" in *Progress Developer Studio for OpenEdge Online Help*.
6. After the instance starts, send a request to the instance. Progress Developer Studio switches to the Debug Perspective when a breakpoint is found.

For more information, see "Debug an ABL application deployed on a Progress Application Server for OpenEdge" in *OpenEdge Videos*.

About session and request states

Session and request states hold context information inside of the session manager. The session state maintains the context of a logical connection with a client. The request state maintains the context of a running request inside a given session.

The session manager must acquire a connection to a multi-session agent and an ABL session in order to run a request. Multi-session agents, however, do not acquire connections. When started, multi-session agents are generally always waiting for requests.

Session states

A session state is maintained by the session manager. The session state can be viewed in OpenEdge Management or through the session information REST API. The session states can be displayed as:

- INIT / STARTING–Indicates that the session is being initialized by the multi-session agent.
- AVAILABLE–Indicates that a session is available for use. A session cannot be acted upon simultaneously by multiple threads.
- RESERVED–Indicates that a session is running an active request and is unavailable to other processes.
- STOPPING–Indicates that the session is awaiting the stopping processes to complete on the multi-session agent.
- STOPPED–Indicates that the session has stopped. This state is displayed before the session is trimmed.

Request states

A request state indicates the messaging status between the session manager and the multi-session agent. The request state can be viewed in OpenEdge Management or through the current request information REST API. The request states can be displayed as:

- `INIT`–Indicates the initiation of the request object in the session manager.
- `READY`–Indicates that the multi-session agent has acquired a connection and ABL session necessary to run a request.
- `WRITING`–Indicates that the client is sending packets of the request to the multi-session agent.
- `RUNNING`–Indicates that the final packet of the client request was sent to the multi-session agent, but that the multi-session agent has not yet sent the first packet of the response.
- `READING`–Indicates that the client request is in the process of receiving a response from the multi-session agent.
- `DISCONNECT`–Indicates that the request is running a disconnect procedure.

For more information about viewing session and request states in OpenEdge Management or OpenEdge Explorer, see View OpenEdge ABL application client sessions in *Manage Progress Application Server (PAS) for OpenEdge with OpenEdge Management*.

For more information about viewing session and request states using the OpenEdge Manager REST APIs, see Get session information and Get current request information in *PAS for OpenEdge Administration Tools Reference*.

Troubleshoot problems with an instance

When you debug an issue with PAS for OpenEdge, there are several areas to consider:

1. Tomcat layer—Handles the incoming messages
2. Session Manager layer—Manages incoming messages to the multi-session agent
3. Multi-session agent layer—Handles ABL requests

Methodology for troubleshooting

Because the majority of activity occurs within the agent, the common place to begin investigating a problem is within the *`abl-app-name`*`.agent.`*`date`*`.log` file.

1. Open the *`abl-app-name`*`.agent.`*`date`*`.log` file.
2. Review the file for the time of the error. The time is based on the system time of the PAS for OpenEdge Tomcat process.
3. If no errors are listed, then go up one level to the *`abl-app-name`*`.`*`date`*`.log` file. For more information about ABL application logging, see Configure ABL application logging on page 230.
4. Review the file for the time of the error.
5. If no errors are listed within these files at the expected time, then go up another level to see if an error is recorded in one of these log files:
   - `catalina.`*`date`*`.log`
   - `localhost.`*`date`*`.log`
   - `localhost-access.`*`date`*`.log`

6. If the error cannot be found in these log files, change the verbosity of your log output by setting the logging levels in `logging-pasoe.properties` and `logging-tomcat.properties`. See Configure logging in PAS for OpenEdge on page 224.
7. You can specify additional Log Entry Types in `openedge.properties` for your agent:
   - `DB.Connects`
   - `DynObjects.DB` (You can log all DynObjects information by using `DynObjects.*`)
   - `DynObjects.XML`
   - `DynObjects.Other`
   - `FileID`
   - `QryInfo`
   - `ProEvents.UI.Char`
   - `ProEvents.UI.Command`
   - `ProEvents.Other`
   - `SAX`
   - `UBNet`
   - `MSASSessions`
   - `MSASSignals`
   - `SEC.AUTH`

   For more information, see "Log Entry Types" in *Startup Command and Parameter Reference*.
8. If memory consumption for the multi-session agent is continually increasing, there may be memory leaks in the application code. For more information, see Find memory leaks using ABL object tracking on page 293.
9. If a severe error occurs on a multi-session agent, then a graceful emergency shutdown is initiated to avoid further failures. For more information, see Graceful emergency shutdown of a multi-session agent on page 292.

Graceful emergency shutdown of a multi-session agent

If a severe error occurs, then a graceful emergency shutdown is initiated. The shutdown process causes the multi-session agent to redirect all new requests to a new multi-session agent and records a message in the Syslog (UNIX) or Eventslog (Windows) file and then generates Protrace. The shutdown process also creates a new file, `pasoe-agent-crash-trace-`*`process-id`*. The Protrace and `pasoe-agent-crash-trace-`*`process-id`* files are placed in the `$DLC/WRK/`*`InstanceName`*`/work` directory.

The `pasoe-agent-crash-trace-`*`process-id`* file collects the deferred logs and ABL call stacks of other running requests and also captures all the other information to help debug the root cause of the failure.

When the emergency shutdown process initiates, the multi-session agent waits to complete any of the active requests. The default wait time is 600000 milliseconds (10 minutes). You can customize the wait time by updating the `completeActiveReqTimeout=`*`600000`* parameter in the `openedge.properties` file.

Find memory leaks using ABL object tracking

The ABL object tracking APIs can be used to find memory leaks in ABL code on a PAS for OpenEdge instance.

The ABL object tracking APIs are an improvement over the previous method of tracking ABL objects, which required enabling the `DynObjects` log entry types and creating a `LOG-MANAGER` (`-clientlog`) system handle. This produced a complex log file that had to be manually searched for memory leaks.

The benefits of using the ABL object tracking APIs to find memory leaks in PAS for OpenEdge are that:

- the functionality can be turned on and off interactively, so there is no reason to shut down a production application to make logging changes.
- the APIs only show memory leaks, rather than making additional calls to the log file for Dynamic Objects.
- the APIs result in less (if any) server slow-down than in the previous method of using Dynamic Objects, and can be turn off interactively to limit performance costs.

To use ABL object tracking to identify memory leaks in a running PAS for OpenEdge instance:

1. Enable ABL object tracking:
   - Using the REST API Enable or disable ABL object tracking
   - Using OEJMX API Enable or disable ABL object tracking query
2. Verify the ABL object tracking status is enabled:
   - Using the REST API Check ABL object tracking status
   - Using OEJMX API Check ABL object tracking status
3. Continue to let the application operate with ABL object tracking enabled.
4. Generate a report of the tracked ABL objects:
   - Using the REST API Get a report of tracked ABL objects
   - Using OEJMX API Get a report of tracked ABL objects query
5. Disable ABL object tracking after leaks have been identified and resolved:
   - Using the REST API Enable or disable ABL object tracking
   - Using OEJMX API Enable or disable ABL object tracking query

8

Optimize PAS for OpenEdge for continuous operations

Continuous operations means that an OpenEdge application can continue to function during a down-time event. Continued application availability includes ensured data integrity and the availability of the application infrastructure.

PAS for OpenEdge includes functionality that provides consistent application processing:

- Secure online deployment of a new ABL application.
- Update PROPATH in a production instance with zero downtime.
- Refresh agents API so that multi-session agents can process requests from updated applications.
- Enhanced ABL session management on page 298

For details, see the following topics:

- Secure online deployment of a new ABL application
- Update PROPATH in a production instance with zero downtime
- Refresh agents in an ABL application
- Enhanced ABL session management

Secure online deployment of a new ABL application

OpenEdge system administrators can securely deploy new ABL applications to production PAS for OpenEdge instances without having users experience any downtime. A secure ABL application deployment to a production instance requires a securely deployed Tomcat Manager, as well as other security considerations, that are described in this topic.

Note: The best practice is to not deploy applications to an online instance. However, there may be circumstances when this is required. If you enable this capability, ensure that you secure your instance.

To enable secure the online deploment of new ABL applications:

1. **Deploy Tomcat Manager.**

   To deploy ABL applications online, you need to use Tomcat Manger features. For security reasons, Tomcat Manager is not deployed, by default, on production servers.

   To deploy Tomcat Manager:

   ```
   proenv> pasman.[sh|bat] deploy -I instance-name $DLC/extras/manager.war
   ```

2. **Secure the Tomcat Manager remote address valve.**

   To maintain the security of your PAS for OpenEdge instance, do not enable remote administration on a production server. When you enable Tomcat Manager to allow online deployment of ABL applications, limit access to Tomcat Manager to localhost requests. Specify the localhost requests using the remote address valve configuration in *instance-name*`/webapps/manager/META-INF/context.xml`, as shown:

   ```
   <!--
   Remove the comment markers from around the Valve below to limit access to
   the manager application to clients connecting from localhost
   -->
   <!--
   <Valve className="org.apache.catalina.valves.RemoteAddrValve"
   allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
   -->
   ```

   The remote address valve compares the IP address of the client that submitted the request against one or more *regular expressions*, and either allows the request to continue or refuses to process the request from the client.

   Note: With IPv6 addresses, the format of the IP address this valve processes depends on the API that was used to obtain it. If the address was obtained from a Java socket using the `Inet6Address` class, then its format is `x:x:x:x:x:x:x:x`. That is, the IP address for the localhost is `0:0:0:0:0:0:0:1` instead of the more widely used `::1`. Consult your access logs for the actual value.

3. **Employ additional security recommendations.**

   Using Tomcat Manager, or other management applications, requires you to carefully secure your environment. Because the Tomcat Manager application allows for web applications to be deployed remotely, it can be targeted by attackers. This is often due to publicly accessible Tomcat instances that have weak passwords and have the Manager application enabled. If the Manager application is enabled, then follow these security guidelines:

   - Ensure that any users allowed to access the Tomcat Manager application have strong passwords.
   - Change the default username and password.

- Use roles to limit deployment capabilities to authorized users.
- Use `LockOutRealm` (enabled by default) that prevents brute force attacks against user passwords. Do not disable `LockOutRealm`.
- Use file system security to protect the web application configuration, and use the PAS for OpenEdge security configuration files.

4. **Deploy the ABL web application.**

   After Tomcat Manager is enabled and secured, you can use the PASMAN command to deploy an ABL application. When you deploy a web application that requires tailoring to a running PAS for OpenEdge instance, a restart is normally required to load that web application's context. For an online deployment, you can use the `-l` (lowercase L) option to load the context of the application on a running server without restarting the server.

   To deploy an `oeabl.war` based web application:

```
proenv> pasman.[sh|bat] deploy -I instance-name -l -u myuid:mypwd -v
$DLC/extras/oeabl.war ABLapp-name
```

Note: Tomcat Manager (`manager.war`) must already be deployed for the `-l` switch to work. If Tomcat Manager is not deployed, then you are prompted to restart the instance. For production instances, you must deploy the Tomcat Manager before you can use this switch. Also note that the `-l` switch can be used only in conjunction with the `-u` parameter.

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 handling requests begin executing the new application code. PAS for OpenEdge ABL sessions 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`. Bound sessions are not updated with the new PROPATH until the sessions finish executing the bound client requests.

To enable dynamic PROPATH updates in production instances:

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:

```
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.

Refresh agents in an ABL application

The refresh agents feature terminates all sessions within the agent of an ABL application, thereby refreshing the multi-session agents so that changes to the application can be picked up by the agents.

When using this API, the ABL sessions that are running in the application complete before terminating. New sessions are started on the next request to the agent. The refresh agents API allows new sessions to use the updated persistent procedures, static objects, or online schema changes, and start running sessions against the new code.

Because this feature terminates all ABL sessions, it may help reduce memory growth due to application leaks. This may or may not include returning memory to the operating system, but it should at least free up memory currently used in the agents to be reused by new ABL sessions.

The refresh agents API provides the same operations as the Update PROPATH in a production instance with zero downtime on page 297 feature, but does not require a change to the PROPATH or for the runtime updates to be enabled.

The API is accessible through REST, an OEJMX query, and the TCMAN command-line utility. See the following topics for usage information:

- TCMAN reference: "Refresh agents in an ABL application (jmx refreshagents)" in *PAS for OpenEdge Administration Tools Reference*
- OEJMX queries: "Refresh agent query" and "Refresh all agents query" in *PAS for OpenEdge Administration Tools Reference*
- REST API: "Refresh all ABL sessions of one or all agents" in *PAS for OpenEdge Administration Tools Reference*. Refresh agents is also implemented in the Swagger UI. For more information, see Use Swagger UI to explore management REST APIs on page 258.

Enhanced ABL session management

In OpenEdge 12.2, the PAS for OpenEdge multi-session agent self-manages ABL sessions, so that whenever an ABL session creation fails, subsequent requests are queued instead of failing, and executed when an ABL session becomes available.

This addresses failures due to resource unavailability in a production environment and supports continuous operations of ABL applications. The enhanced ABL Session Management functionality allows PAS for OpenEdge to dynamically change its in-memory maxABLSessionsPerAgent based on resource availability.

ABL Session Management reduces the in-memory value of of `maxABLSessionsPerAgent` if the multi-session agent reaches the limit of allowed concurrent connections to a resource. Instead of connections failing due to exceeding the number of concurrent connections to a resource, the multi-session agent alerts the PAS for OpenEdge Session Manager to queue the requests, or start up a new multi-session agent (if one is available), rather than return a failed request.

The value of is `maxABLSessionsPerAgent` is represented in-memory with the property `dynmaxablsessions`.

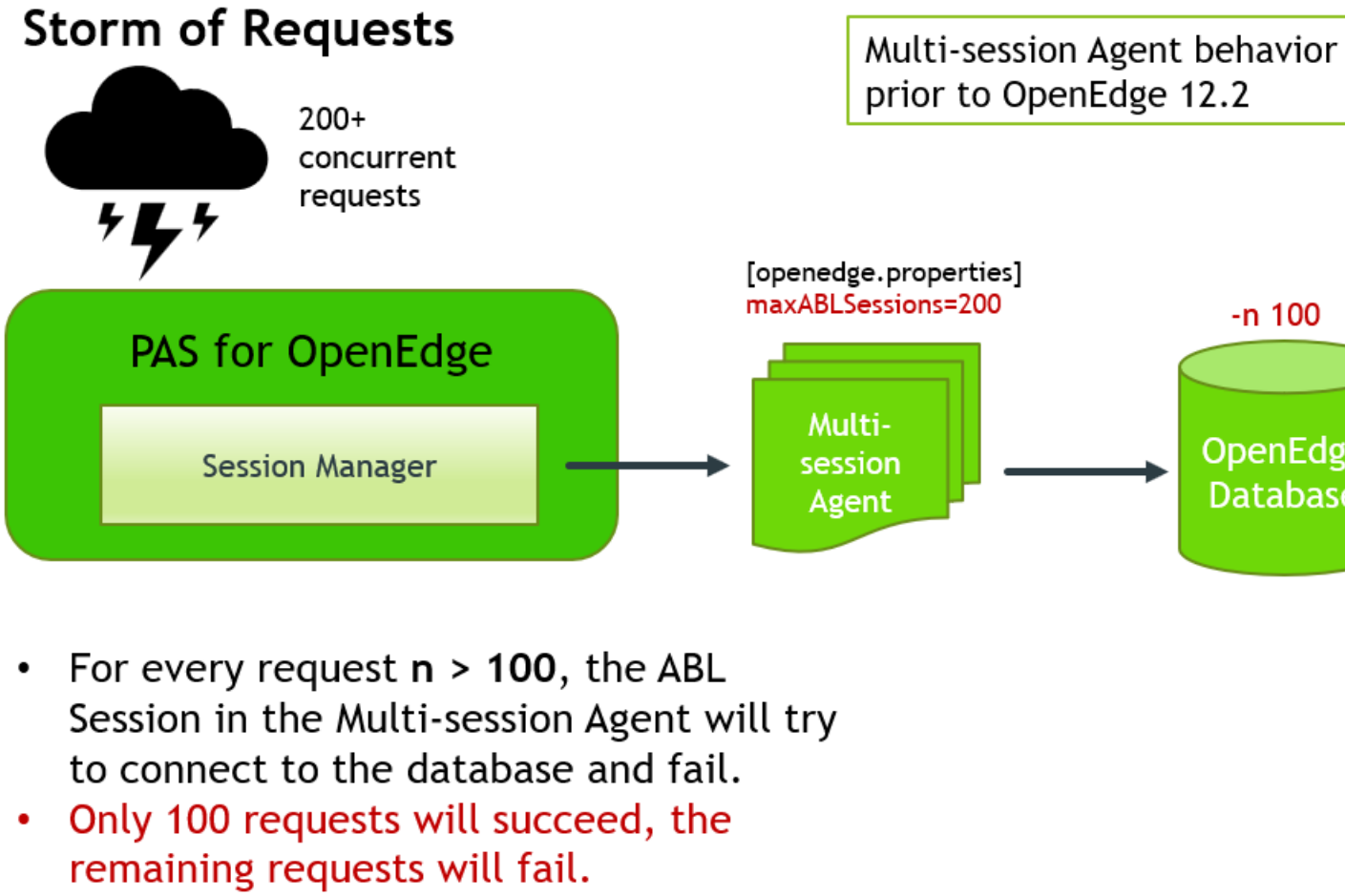
This behavior is enabled by default.

Here's how it works

To provide an example use case:

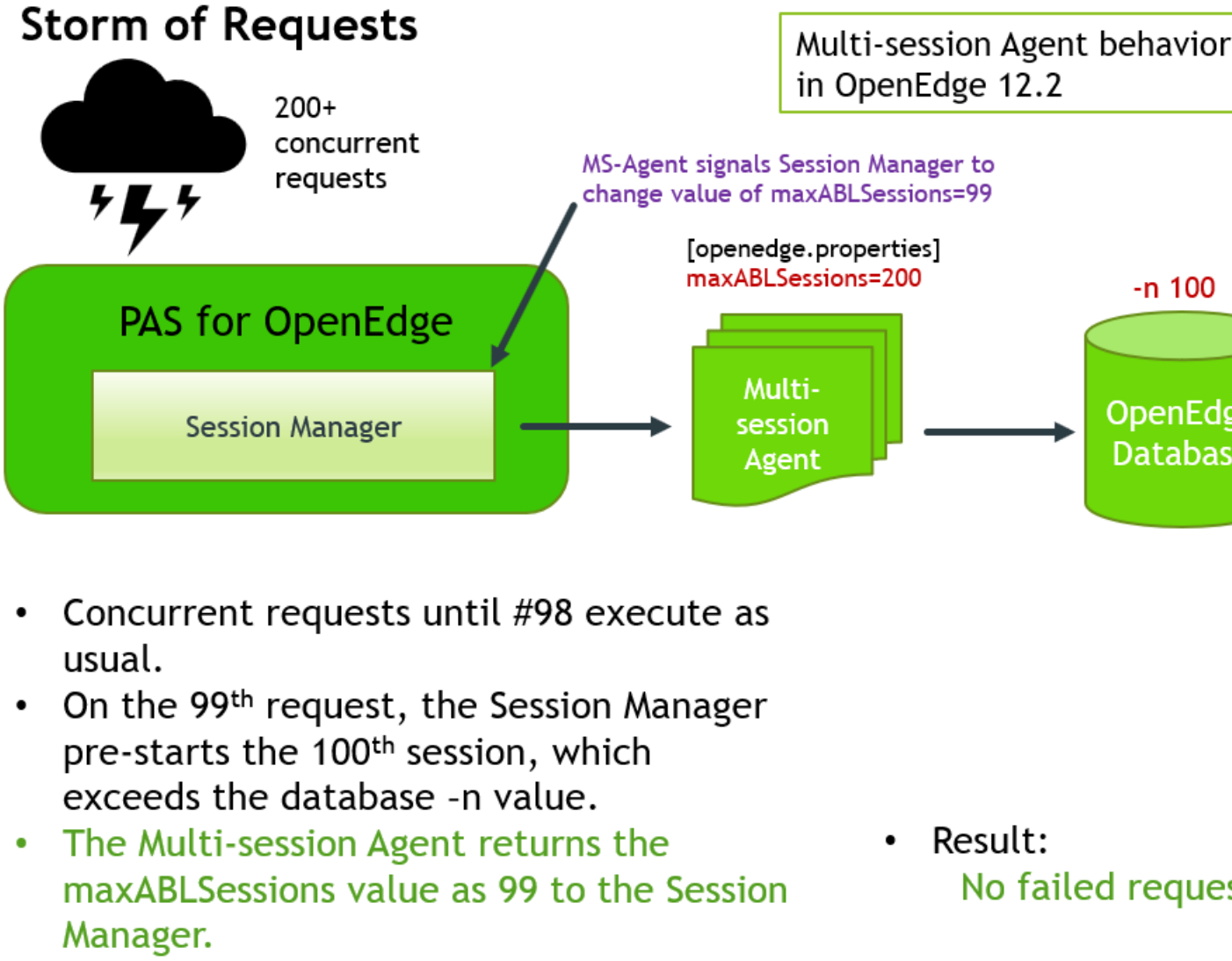
A PAS for OpenEdge instance has its `maxABLSessionsPerAgent` set to 200 in the `openedge.properties` file.

The PAS for OpenEdge instance connects to an OpenEdge database that has a Maximum Number of Users (`-n`) set to 100.



In this example, the database prevents more than 100 concurrent incoming requests, so the multi-session agent cannot start more than 99 ABL sessions when required. Therefore, any additional requested ABL sessions will fail due to exceding resource limits.

If a large connection event (a "storm of requests") occurs where a burst of people send requests, the multi-session agent turns the request storm into new sessions. The multi-session agent then sends the sessions to the OpenEdge database to request a resource. The database rejects concurrent requests after the 100th is received, and then remaining requests return as failed.



With the enhanced ABL Session Management, when the multi-session agent receives the 100th request, the `minAvailableABLSessions` property triggers the multi-session agent to change its in-memory `maxABLSessionsPerAgent` value to 99. This is because the multi-session agent is alerted that there are no more available sessions due to an error message from the database.

With the `maxABLSessionsPerAgent` set to 99 in-memory, the PAS for OpenEdge Session Manager queues the remaining requests until an existing ABL session becomes available. The Session Manager then routes a queued request to an available ABL session in the multi-session agent. Instead of failing, the requests are queued in the Session Manager until a session opens in the multi-session agent.

Controlling Min and Max Available ABL Sessions

The `minAvailableABLSessions` property has been added to `openedge.properties` in support of enhanced ABL Session Management. This property specifies the smallest number of available ABL sessions that are reserved by the multi-session agent during its operation. The `minAvailableABLSessions` property tries to pre-start an ABL session to ensure a resource connection is available. If the multi-session agent detects that the number of available sessions has fallen below the default value of 1, the multi-session agent signals to the Session Manager to lower the `maxABLSessionsPerAgent` to the number of current ABL sessions.

You can set the value of `minAvailableABLSessions` higher than 1 to request that more additional available ABL sessions be reserved for incoming requests.

When the number of `minAvailableABLSessions` falls below the threshold defined in `openedge.properties`, then the value of the `maxABLSessionsPerAgent` property changes to match the number of current ABL sessions.

The change of the `maxABLSessionsPerAgent` value occurs in-memory rather than being written into the `openedge.properties` file. This in-memory ABL session limit is stored in the property `dynmaxablsessions`.

Enhanced ABL Session Management APIs

You can view, update, and reset the in-memory value of the `maxABLSessionsPerAgent` (`dynmaxablsessions`) property using OEJMX queries or the REST APIs provided by the OE Manager application.

OE Manager REST APIs, in *PAS for OpenEdge Administration Tools Reference*:

- "Get dynamic session limit"
- "Update dynamic session limit"
- "Reset dynamic session limit"

OEJMX Queries, in *PAS for OpenEdge Administration Tools Reference*:

- "Get dynamic ABL session limit query"
- "Update dynamic ABL session limit query"
- "Reset dynamic ABL session limit query"

Index

A

Add additional index entries here (if needed)
    Nest index terms for sub entries *36*, *51*
Amazon load balancing *219*
Apache load balancing *219*
Apache proxy host load balancing *212*
Apache proxy load balancing *212–213*
Auto scaling *219*

C

Clusters *210*
CORS support *136*

D

deploy *81*
Development server *91*

E

Elastic load balancing *219*

H

HTTPS *124*

J

JConsole *39*
JMX *39*

L

LDAP security configurations *98*
Load balancing
    Apache proxy host *212*
    Apache proxy load balancing *212*
    Tomcat load balancing *215*

M

MBeans *39*
MemoryRealm *138*

O

OEJMX *43*
OEPROP *34*
OERealm *99*
OpenEdge Explorer *39*
OpenEdge Management *39*

P

PAS for OpenEdge
    development server *91*
    production server *91*
PASMAN *25*
pasoestart *19*
Production server *91*
Progress Application Server
    extensions to Apache Tomcat *13*
Proxy load balancing *214*

R

Realms *137*
Remote access filters *140*
Required: add at least one index entry *36*, *51*
Roles
    assigning to users *138*

S

SSO support *136*
Sticky sessions *210*
stsurlutil
    update *188*

T

TCMAN
    deploy *81*
Tomcat load balancing *215–217*