Progress®



Introduction to Progress Application Server for OpenEdge

OpenEdge®

Copyright

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

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

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

February 2019

Last updated with new content: Release 12.0

Updated: 2019/03/01

Table of Contents

Chapter 1: Introduction to PAS for OpenEdge.....9

PAS for OpenEdge architecture.....10
Instances.....12
ABL applications.....13
The ROOT application.....15
Web applications.....16
Transports.....16
Manager applications.....17
Catalina environment variables.....17
PAS for OpenEdge directories.....17
HTTP sessions.....18
Property files.....18
Log files.....19
Use PAS for OpenEdge with Docker containers.....20
Use PAS for OpenEdge: main steps.....21

Chapter 2: Move a Progress Application Server to Production.....25

Development versus production instances.....26
Missing and disabled features in production.....27
Licensing and installation.....27
PAS for OpenEdge configuration tools.....28
Utilities for managing keys and digital certificates.....28
What you need from the developer.....29
Move to production checklist.....30

Chapter 3: Create and configure a PAS for OpenEdge instance.....31

Create a PAS for OpenEdge production instance.....31
Use PASMAN to create an instance.....32
Use OpenEdge Management to create an instance.....32
Check the security model of the instance.....33
Start and stop a PAS for OpenEdge instance.....34
Use PASMAN to start and stop an instance.....34
Use OpenEdge Management to start and stop an instance.....34
Configure a PAS for OpenEdge instance.....35
Use PASMAN to configure an instance.....35
Use OpenEdge Management to configure an instance.....36
Connect to an OpenEdge database.....37
Use command-line utilities to connect to an OpenEdge database.....38

Use OpenEdge Management to connect to the database....40
Register a PAS for OpenEdge instance as a Windows service....40
Delete a PAS for OpenEdge instance....41
Use PASMAN to delete an instance....41
Use OpenEdge Management to delete an instance....42

Chapter 4: Application deployment in PAS for OpenEdge....43

Configure properties for deployment....44
Set the PROPATH....44
Set database connections....44
Enable transports....45
Set agents and sessions....45
Modify environment variables....46
Stage r-code....46
Deployment artifacts....47
Deploy ABL services....48
Deploy REST services....48
Deploy SOAP services....48
Deploy WebHandler services....49
Deploy an ABL web application....49
Deploy remote administration applications....50
Package an instance for production....50
Update PROPATH in a production instance with zero downtime....51

Chapter 5: Secure a PAS for OpenEdge Instance....53

Production instance security customizations....54
Enable ABL application authentication....55
Disable HTTP in production instances....58
Use SSL/TLS in PAS for OpenEdge....58
Obtain a certificate from a Certificate Authority....59
Configure a PAS for OpenEdge instance for SSL/TLS....62
Configure clients for SSL/TLS....63
Configure Spring Security....64
Single sign-on (SSO) support....64
CORS support....65
Use CORS....66
Manage users and remote access....66
User and file permissions....66
Role-based user authentication....66
Secure online deployment of a new ABL application....68
Remote access filters....70

Chapter 6: Tune PAS for OpenEdge instances....71

PAS for OpenEdge tuning recommendations....72
Goals and common steps for tuning PAS for OpenEdge instances....74
PAS for OpenEdge server-side tuning....75
JVM parameters....75
Network connections....77
HTTP/HTTPS request processing....78
PAS for OpenEdge instance parameters....79
Tune OpenEdge web applications and MS-agents....81
ABL Session Manager and session pool....81
Web application startup parameters....84

Chapter 7: Load balance PAS for OpenEdge instances....87

Scaling considerations in PAS for OpenEdge....87
Enable HTTP sessions for the APSV transport....88
Apache proxy host load balancing....89
Requirements for Apache proxy load balancing....89
Configuring the Apache server for proxy load balancing....90
Configuring PAS for OpenEdge instances for proxy load balancing....91
Tomcat load balancing....92
Requirements for Tomcat load balancing....93
Configuring an Apache web server for Tomcat load balancing....93
Configuring PAS for OpenEdge for Tomcat load balancing....94
Amazon load balancing....95
Configuring Elastic Load Balancing and Auto Scaling....95
Docker container load balancing....98

Chapter 8: Monitor PAS for OpenEdge instances....99

Configure logging in PAS for OpenEdge....100
Configure Tomcat and Tomcat web application logging....100
Configure PAS for OpenEdge application logging....102
Configure log file patterns....107
Configure MDC access for header requests....112
Enable Debugging....113
Use OEJMX to manage and monitor an instance....114
List available MBeans to locate AgentManager details....114
Write a custom OEJMX query to get agents....115
Get agent information....116
Generate a stack trace 116
Common OEJMX queries....118
OEJMX utility....125
Use Server-Side ABL Performance Profiling....126
What is Server-Side Profiling?....126
Configure the diagnostic store....127

Configure Server-Side Profiling with PAS for OpenEdge....128
Supported JSON values for JMX queries....131
Use Swagger UI to explore management REST APIs....133
Enable Swagger UI for management REST API access....133
Explore and test management REST APIs....133
Use the OpenEdge HealthScanner....134
Primary use case....135
How is the data collected?....136
Create an instance....137
Start the HealthScanner....137
Access HealthScanner data....140
Display results of a healthy REST call....149
Display results of an unhealthy REST call....152
Customize HealthScanner reports....155
Use the log file viewer from OpenEdge Management....156
Troubleshoot problems with an instance....157

1

Introduction to PAS for OpenEdge

Progress Application Server for OpenEdge (PAS for OpenEdge) is an Apache Tomcat web server that has been extended to support OpenEdge and ABL. PAS for OpenEdge has built-in routing and adapter functionality that enables you to access your business logic on a variety of clients including ABL, browser-based clients, and mobile clients. Progress development applies Apache Tomcat updates that ensure compliance with industry standards, like Spring security, while supporting all client types in a single server using easy to configure properties files. Because PAS for OpenEdge is a web server, it makes implementing and deploying applications efficient and simplifies application management.

To use PAS for OpenEdge, you:

- Create and configure instances.
- Deploy ABL or Webspeed application code to an instance.
- Deploy REST and SOAP services to an instance.
- Create and deploy Web handlers for application modernization.
- Start, stop, and delete instances.
- Deploy management applications such as Tomcat manager and OpenEdge manager to an instance.
- Monitor and manage instances.
- Implement load balancing.
- Implement security such as HTTPS, user authentication and authorization.

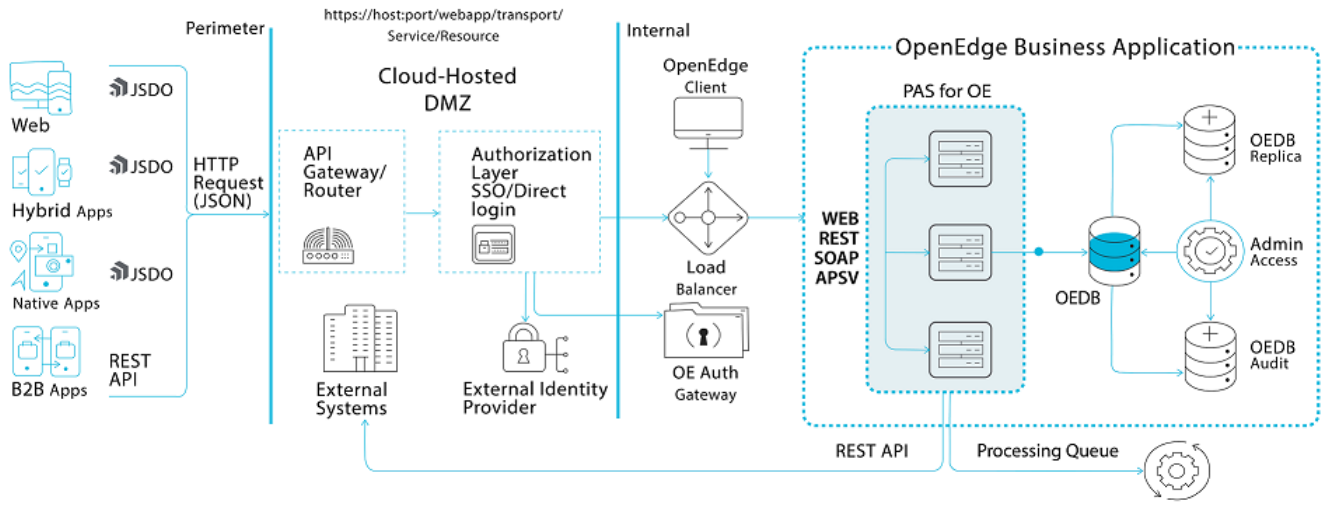
For details, see the following topics:

- PAS for OpenEdge architecture

- Instances
- ABL applications
- The ROOT application
- Web applications
- Transports
- Manager applications
- Catalina environment variables
- PAS for OpenEdge directories
- HTTP sessions
- Property files
- Log files
- Use PAS for OpenEdge with Docker containers
- Use PAS for OpenEdge: main steps

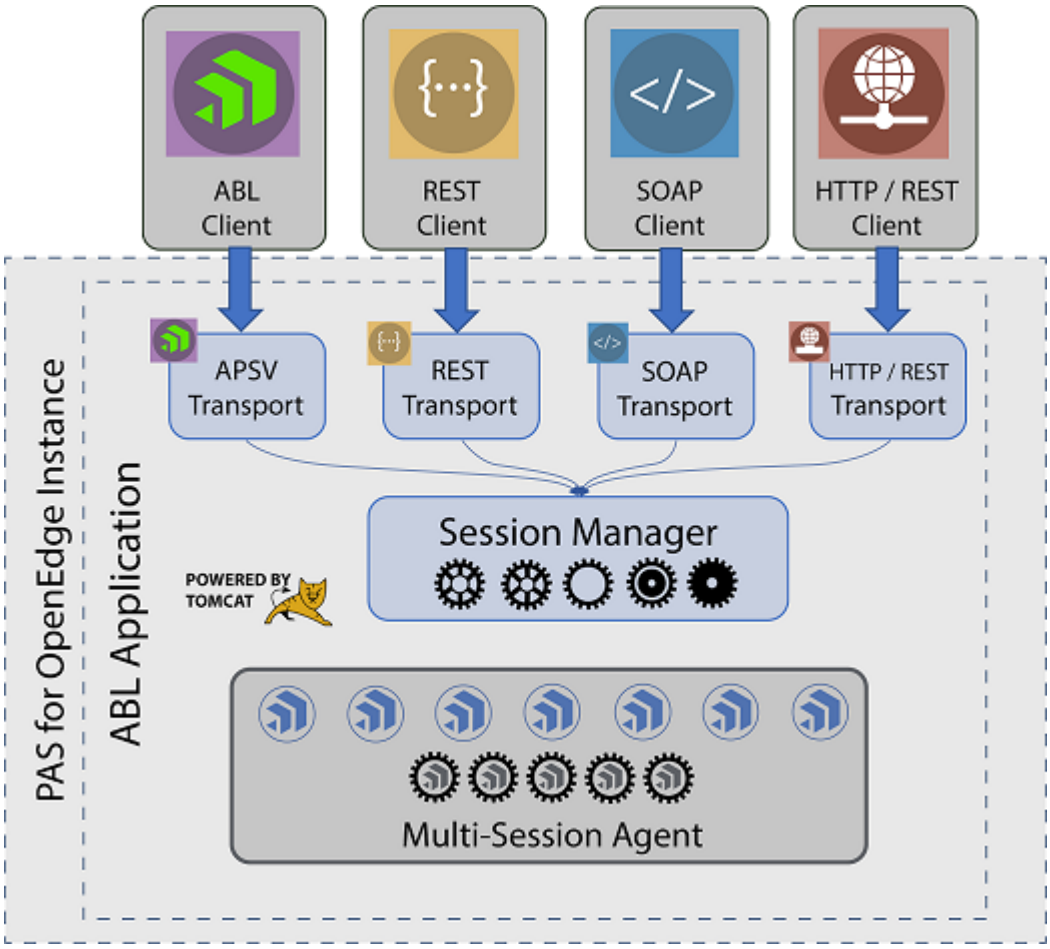
PAS for OpenEdge architecture

PAS for OpenEdge is a component of the OpenEdge environment. The following illustration shows PAS for OpenEdge and related OpenEdge components:



PAS for OpenEdge is the web server component of an OpenEdge Business Application. PAS for OpenEdge interacts with the OpenEdge database and serves client requests using transports.

Each PAS for OpenEdge instance has the following components, which are shown in the following illustration:



This figure shows the following:

- An Apache Tomcat web server.
- Transports—Supporting the following types of client connections (APSV for ABL, WEB for HTTP-based clients or new REST applications, REST and SOAP for HTTP/S).
- Session Manager—A process that manages pools of ABL sessions, where each pool runs in the context of a multi-session agent. The Session Manager coordinates requests from clients and locates ABL sessions to handle each request.
- Multi-Session Agent (MS-agent)—A pool of ABL sessions that executes ABL business logic on behalf of clients.

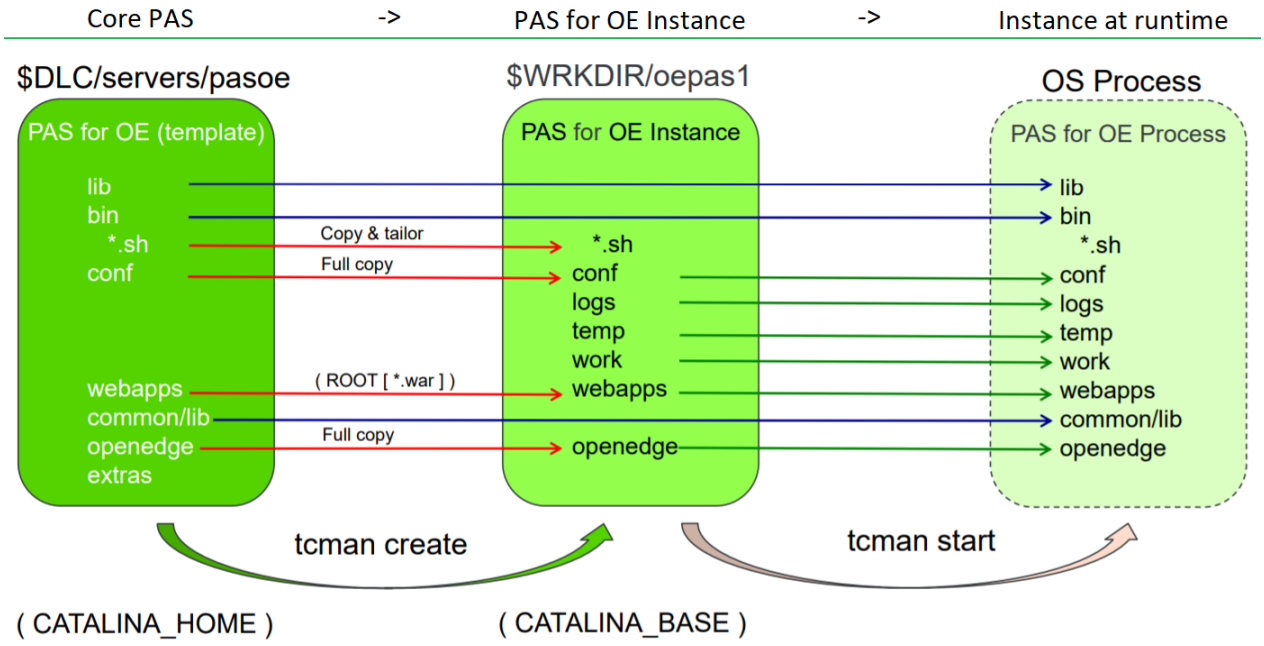
A PAS for OpenEdge instance can have one or more ABL applications. Each ABL application can have:

- Its own security, startup, and propath configurations within the same instance
- One or more web applications derived from `oeabl.war`
- One or more ABL services. Each ABL service is represented by a `.paar` file (REST transport), `.wsm` file (SOAP transport), or a Web handler (WEB transport).

Instances

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

The following image illustrates the relationship between the core PAS and any PAS for OpenEdge instance that is created:



When you create an instance, the root directory of the instance is assigned to the `CATALINA_BASE` environment variable within the scripts in the *`instance-path`*`/bin` directory. The root directory of the PAS installation is assigned to the `CATALINA_HOME` environment variable in the scripts in the *`instance-path`*`/bin` directory. The scope of these environment variables is limited to the context of an individual *`instance-path`*`/bin` scripts.

Progress requires that you deploy your web applications to an instance of PAS for OpenEdge, rather than deploying to the PAS for OpenEdge installation directory. This practice prevents accidental corruption of the core executables, configuration settings, and libraries. It also prevents accidental deletion of web applications if the core Progress Application Server is removed during an uninstall.

Some advantages of instances are:

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

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

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

Note: Catalina is the Apache product name of the component that implements the Java Servlet Container for Tomcat.

The structure of a PAS for OpenEdge instance (which is identified by an alias name and its OS file system path) includes the following directories:

- **`bin`** — A directory that contains OpenEdge and customer-written utilities and libraries that are tailored to that specific instance. For example, OpenEdge utilities are tailored to use the local file space of the instance but load and use core (in other words, `$CATALINA_HOME`) shared libraries and utilities.
- **`common/lib`** — A directory that contains java libraries and files that are shared across all of the Web applications of the instance

  ---

  Note: The core server also has a `common/lib` directory that holds java libraries and files shared by all Web applications across all instances.

- **`conf`** — A directory that contains the Tomcat, OpenEdge, and customer-written configuration files of the instance that control only the applications deployed into the instance
- **`logs`** — A directory that contains the log files for the server instance, including all of the server, Web application, OpenEdge, and ABL application log files.
- **`work`** — A directory that contains ABL web application generated persistent working files. These files should not be deleted by an administrator as part of cleaning up before doing an application restart.
- **`temp`** — A directory that contains ABL web application generated temporary files. These files can be deleted by an administrator as part of cleaning up before restarting the server instance.
- **`webapps`** | ***`webapps-ref`*** — A directory, or a reference to a directory, that contains the Tomcat, OpenEdge, and customer ABL Web applications of the instance.
- **`openedge`** — A directory that contains `.p` and/or `.r` files and supporting files only for ABL applications. Other Progress and/or third party Web applications cannot write or read from this directory.

See also:

- PAS for OpenEdge directories

ABL applications

A PAS for OpenEdge instance and the one or more ABL applications that it hosts have a defined structure with specific locations for different components.

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

- A unique name which can be referenced from administrative tools

- A set of r-code files that comprise the application
- A single PROPATH value that is used to access the r-code of the application
- One or more OpenEdge database connections
- A set of start-up options

As an example of an ABL application, the OpenEdge ABL web application (`oeabl.war`) is structured as follows:

- **`/apsv`** — A URI path reserved for OpenEdge client connections (can be disabled)
- **`/soap`** — A URI path reserved for OpenEdge soap client connections (can be disabled)
- **`/rest`** — A URI path reserved for OpenEdge REST clients (can be disabled)
- **`static`** — A directory that contains static files, including images, style sheets, and HTML pages.
- **`META-INF`** — A required directory containing metadata and `context.xml` definitions specific to the web application. OpenEdge uses a tailored `context.xml` file for the web application, and can be tailored by the deployment site to meet security requirements
- **`WEB-INF`** — A required directory holding the configuration file of the web application (`web.xml`), Spring Security configuration files, and other application private files
  - **`adapters`** — OpenEdge directory tree holding REST and SOAP deployment files
  - **`lib`** — Required directory that holds any Java libraries specific to the Web application
  - **`classes`** — Required directory that holds any Java classes or data files specific to the Web application
  - **`openedge`** — Directory provided intended for ABL developer to distribute their r-code files that apply specifically to the Web application

As you can see, there are specific places where OpenEdge recommends that ABL application developers locate their files. While an ABL application can physically locate its files anywhere in the OS file system, locating them inside the PAS for OpenEdge web application instance promotes scaling, extensibility, and deployment. To promote scalability, extensibility, and deployment, a PAS for OpenEdge ABL web application should contain the following components:

- An ABL application (identified by name) that may be deployed into an existing PAS for OpenEdge instance.
- Common `.r` or `.p` files used across multiple web applications located in the instance `openedge/*` directories. The directory structure should be capable of being configured read-only for a secure installation.
- One or more Web applications like `oeabl.war` that may be deployed to an existing PAS for OpenEdge instance. They are used to host APSV, SOAP, and REST transport access to ABL business logic.

  For each web application, include:

  - (Optional) The `.r` and/or `.p` files specific to the application. They should be located in a web application `WEB-INF/openedge` directories. The directory structure should be capable of being configured read-only in a secure installation
  - Static pages used by the Web application. They should be located from the root URI (/) , but not from one of the reserved OpenEdge URIs.
  - Configuration files for the web application, including the application specific `web.xml` files and `oeablSecurity*.xml` files.
  - A default REST configuration properties file.

Note: REST properties are not contained in the `conf/openedge.properties` file.

- Zero or more REST `.paar` files that may be incrementally deployed into an existing OE ABL Web application.
- Zero or more SOAP `.wsm` files that may be incrementally deployed into an existing OE ABL Web application.
- A fragment of the `openedge.properties` file that contains the defaults for the application and its associated web applications, transports, session manager, and multi-session agents. The fragment is appended to the `conf/openedge.properties` file at installation time by a user-written tailoring script.
- An optional user-written installation tailoring script that customizes Web applications and `openedge.properties` files.
- An optional user-written startup environment script (*myapp*`_setenv.` { `sh` | `bat` }) that sets OS process environment variables used by the ABL application code.

The ROOT application

Apache Tomcat comes configured with a number of web applications, which are reconfigured in PAS for OpenEdge. Tomcat requires that there always be a ROOT application.

In PAS for OpenEdge, the default ROOT application is a deployment of `oeabl.war`, the OpenEdge ABL web application. The file `oeabl.war` is deployed with every PAS for OpenEdge instance. You can undeploy ROOT, and deploy any valid web application in its place. The OpenEdge ABL web application is a type of Java web application that is preconfigured with security and Java Servlet interfaces that translate HTTP client requests into ABL requests, and schedules their execution using the ABL Session Manager.

Note: The Tomcat application server requires a web application in the ROOT directory. In the PAS for OpenEdge implementation, the Tomcat application server ROOT web application is replaced by the OpenEdge ABL web application. The ROOT web application in a PAS for OpenEdge instance is the `oeabl.war` file deployed with the name ROOT, not the `ROOT.war` file for the Tomcat server. That file and other standard Tomcat configuration files and applications can be found in `$CATALINA_HOME/extras`.

When you create a PAS for OpeEdge instance, an OpenEdge ABL web application,`oeabl.war`, is automatically deployed in the *instance-path*`/webapps/ROOT` directory. The ABL web application supports all transports for an ABL application, including SOAP and REST web services, Web handlers, ABL and Open Client (APSV) access. In addition, an ABL application itself can be represented by multiple ABL web applications deployed to a single PAS for OpenEdge instance.

The `ROOT` directory contains the manifest file for `oeabl.war` (in the `META-INF` folder), static files (in the `static` folder), the application definitions (in the `web.xml` file) and other pertinent resources such as classes, transports (APSV, REST, SOAP, and Web), security files, and the default PROPATH folder.

Web applications

The web applications deployed to a PAS for OpenEdge instance are generated as web application archive (WAR) files. When deployed, WAR files are expanded in the Java Servlet container of the server, usually in the `/webapps` directory of the server.

In PAS for OpenEdge, the `/webapps` folder typically includes:

- A PAS for OpenEdge ROOT application that replaces the default Tomcat ROOT application.

  Note: The server must have an application named ROOT installed.

- Optional administrative applications such as `manager.war` and `oemanager.war`.
- The ABL web applications that the application server runs in response to a client request.

A deployed web application is represented by a `Context` container in the `/conf/server.xml` file.

Transports

In PAS for OpenEdge, transports handle HTTP requests and are built into the default ROOT (`oeabl.war`) application. Transports handle requests from clients that use any one of those protocols for communication with a web server.

The supported transports and their assigned URLs are described in the following table:

Transport	Description	URL Path
APSV	Handles ABL clients, Java and .NET Open Clients requests. This transport also provides backwards compatibility for Classic AppServer clients over HTTP protocol.	`/apsv`
REST	Handles REST requests.	`/rest`
SOAP	Handles SOAP 1.1 requests.	`/soap`
WEB	Handles REST, classic WebSpeed, and HTTP requests. Offers the option to modernize your application with WebHandlers.	`/web`

Note: Because a PAS for OpenEdge instance can contain one or more ABL web applications, if you are referencing an application other than the ROOT application, the URL path is preceded by the application name (for example, *`webapp_name`*`/web`).

Manager applications

In PAS for OpenEdge, the OpenEdge Manager application, `oemanager.war`, is deployed to manage instances. OpenEdge Manager is a Java web application that provides a REST API for remote administration of the OpenEdge web applications and multi-session agents. It duplicates the administration API supported by JMX interface from Tomcat, but it uses JSON input/output payloads instead. The `oemanager.war` file is required to use REST APIs to manage and monitor of ABL applications and PAS for OpenEdge instances.

Tomcat Manager is deployed to PAS for OpenEdge using `manager.war`.

Note: Although they may not be deployed in PAS for OpenEdge, standard Tomcat configuration files and applications (like `host-manager.war`) are preserved in `$CATALINA_HOME/extras`.

Also be aware that for security reasons OpenEdge Manager may not be deployed or may be undeployed on a PAS for OpenEdge instance if, for example, you don't want remote administration enabled on a production server instance. Tomcat Manager and OpenEdge Manager are deployed on a development instance by default and are required for integration with development tools. Some OpenEdge TCMAN commands, such as `tcman.sh info`, require Tomcat Manager to be loaded and running to work.

Catalina environment variables

There are a number of environment variables that define values for some of the key directories in PAS for OpenEdge:

- `CATALINA_HOME`: The root directory where you installed PAS for OpenEdge. The installed server is often referred to as the core PAS. The `CATALINA_HOME/bin` directory contains all of the PAS for OpenEdge executables, including `catalina.{sh|bat}`, which is a script that is responsible for starting and stopping the core server.
- `CATALINA_BASE`: The root directory of a runtime instance of the core PAS for OpenEdge. If you have a number of instances, the value of `CATALINA_BASE` differs for each individual instance. However, the value of `CATALINA_HOME` is the same for all instances of a general core server.
- `CATALINA_TMPDIR`: The temporary file directory of the core PAS for OpenEdge.
- `CATALINA_PID`: The path of the file holding the process ID for the Catalina Java startup process. Unlike Tomcat, PAS for OpenEdge enables the use for this variable in both UNIX and Windows systems.

PAS for OpenEdge directories

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

Table 1: Progress Application Server directory structure extensions

`$CATALINA_HOME/common/lib/`	Contains general third-party libraries that are shared by a server, its instances, and its web applications.

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

HTTP sessions

HTTP sessions are an industry standard feature that allow web servers to maintain user identity and store user-specific data during multiple request/response interactions between a client application and a web application.

HTTP sessions preserve:

- Information about the session itself (session identifier, creation time, time last accessed)
- Contextual information about the user (client login state, and other user information the web application needs to save)

This context information is required for load balancing. While SOAP, REST and WEB transports use HTTP sessions implicitly, the APSV transport adds this functionality through a combination of properties that are enabled by default for the ROOT application.

Property files

When you create a PAS for OpenEdge instance, the properties of the instance are automatically set to certain default values. Every PAS for OpenEdge instance has its own /conf directory that contains multiple configuration files, including:

- openedge.properties—Contains the properties that must be set to create an instance. See the openedge.properties.README for more information.
- catalina.properties—Contains the Java properties used by an instance and its ABL Web applications. See the catalina.properties.README for more information.
- appserver.properties—This is linked to the catalina.properties file such that any changes made to catalina.properties are dynamically reflected in appserver.properties. See the appserver.properties.README for more information.
- jvm.properties—Contains a list of JVM startup command-line options.
- server.xml—Contains the server features for the instance.

Note: Progress recommends you do not edit server.xml.

You can edit the property files using the PASMAN and OEPROP utilities, as well as OpenEdge Management.

Log files

PAS for OpenEdge creates log files at three distinct levels:

- **Tomcat logs** contain logging information for the web server and JSP servlet container of a particular PAS for OpenEdge instance. These logs include Catalina logs for the core server, localhost logs for the Tomcat virtual server, server access logs, and Tomcat manager logs.
- **ABL application-level logs** contain logging information for ABL applications running on a PAS for OpenEdge instance. This includes Spring Security authentication and authorization process event logs and multi-session agent logs containing logging information for the ABL Session Manager running on a PAS for OpenEdge instance.
- **Instance-level independent logs** include remote management and server support logs, as well as user-deployed Java web applications.

PAS for OpenEdge logs and their contents

The tables in this section list the log files produced by each component of PAS for OpenEdge, and the kind information to be found in each log file.

The default location for all these log files is `$CATALINA_BASE/logs`.

The `$CATALINA_BASE` environment variable defines the root directory of a PAS for OpenEdge instance. On a UNIX-based operating system, reference the logs directory as `$CATALINA_BASE/logs`. On Windows, reference `%CATALINA_BASE%\logs`.

Table 2: Tomcat logs

Filename	Description
`catalina.`*`date`*`.log`	Tomcat server or any of the deployed web application activity log.
`catalina.out`	A UNIX-only catch-all log file that contains everything written to Tomcat console output for every application or process.
`localhost.`*`date`*`.log`	Web application activity. The default Tomcat host name log.
`localhost-access.`*`date`*`.log`	Requests processed by the web server.
`manager.`*`date`*`.log`	Activity of the Tomcat manager.
`host-manager.`*`date`*`.log`	Activity of the Tomcat manager for virtual hosts.

Table 3: Multi-session agent logs

Filename	Description
instance-name.agent.log	Activity of ABL sessions. Similar to the server log of a Classic AppServer.

Table 4: ABL application logs

Filename	Description
ABL-app-name.*date*.log	Lists each interaction that the ABL application has with the agent. Similar to the broker log of a Classic AppServer.
ABL-app-name.agent.*date*.log	The ABL application agent process log output.
*ABL-app-name*_authn.*date*.log	The log file contains the audit trail of failed and passed Spring Security authentication process events. The log is shared by all deployed web apps and transports mapped to the ABL application. This log file can help detect possible server attacks, and track users who access the application.
*ABL-app-name*_authz.*date*.log	The log file contains the audit trail of failed and passed Spring Security URL access control process events. The log is shared by all deployed web applications and transports mapped to the ABL application. This log file can help detect possible intruder scanning.
oests.*date*.log	The Spring Security and Security Token Service web application log that is found only in the OpenEdge Authentication Gateway server.

Table 5: Independent web application logs

Filename	Description
oemanager.*date*.log	The log file of the remote ABL application administration API.
oehealth.*date*.log	The log file of the OpenEdge HealthScanner web application.
oedbg.*date*.log	The log file of the OpenEdge Debugger web application.

Use PAS for OpenEdge with Docker containers

You can now run PAS for OpenEdge instances in a Docker container. This feature gives you the capability to perform zero down-time versioning of PAS for OpenEdge instances.

Docker is a technology that allows developers to develop, deploy, and run an application in a standalone, virtualized environment called a container. Containers are a lightweight, standardized, and secure method of executing software in an isolated environment. Using containers to isolate applications allows for faster and easier development and deployment.

Virtual machines offer a similar benefit to containers, but containers have the advantage because they do not virtualize hardware. Containers work on the application layer, and multiple containers share the same operating system kernel. This technology makes running multiple containers much more portable and efficient than running multiple virtual machines.

Continuous integration and continuous deployment (CI/CD) is one of the biggest benefits of using Docker containers. CI/CD allows developers to build, test, and deploy code in a wide variety of environments and allows them to test as frequently as they desire. By combining the development and testing processes, CI/CD and Docker containers can make developing, testing, and deploying software much more efficient.

Containers provide several advantages when used with PAS for OpenEdge:

- Using Docker containers helps reduce overhead for packaging and deploying ABL applications, allowing you to streamline your application distribution processes. You only need to create a Docker custom container image with PAS for OpenEdge once, then you can distribute that image and run as many containers as needed.
- Similarly, when upgrading your Docker containers for PAS for OpenEdge, you only need to upgrade one image, and then you can easily redistribute the upgraded image.
- Clustering tools such as Docker Swarm and Kubernetes allow you to run multiple PAS for OpenEdge containers in a cluster, which provides high availability and elasticity.

For more information on containers, see Run PAS for OpenEdge in a Docker Container.

Use PAS for OpenEdge: main steps

To use PAS for OpenEdge in their environments, developers and system administrators perform the following steps.

Developer

As a OpenEdge developer, your responsibilities for using PAS for OpenEdge include the following:

1. **Install Progress Developer Studio.**

   A development instance of PAS for OpenEdge is installed as part of Progress Developer Studio.

2. **Create a development instance.**

   Use Progress Developer Studio to create new instances, or use the default development instance, `oepas1`.

3. **Define properties for your instance.**

   Use the Progress OpenEdge options in the Preferences menu to set launch configuration, startup, server, propath and database connection properties.

4. **Define security.**

   Configure the Spring Framework and authentication mechanisms.

5. **Debug your development instance.**

   Use Debug view in Progress Developer Studio.

6. **Create archives for production.**

   Package your instance for deployment.

System Administrator

As a system administrator for PAS for OpenEdge, your responsibilities include the following steps to bring your server online:

1. **Move to a PAS for OpenEdge production instance.**

   - **Install the license for a production instance.**

     Obtain a license for and install the production version of PAS for OpenEdge.

   - **Create a production instance.**

     Use PASMAN or OpenEdge Management to generate a production instance.

   - **Configure a production instance.**

     Use PASMAN or OpenEdge Management to set server properties, enable transports, and (optionally) deploy the WebSpeed handler.

   - **Deploy applications to production instance.**

     Use PASMAN or OpenEdge Management to deploy applications to your instance.

2. **Secure your production instance.**

   - Get certificates and keys from IT.
   - Enable SSL.
   - Check user and file permissions.
   - Set authentication level for instance and applications.

3. **Test and monitor production instances.**

   - Monitor your production instance using OEJMX or REST APIs.

- Tune your instance.
- Implement load balancing.

2

Move a Progress Application Server to Production

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 two products with different licensing:

- A development mode product: Progress Development Application Server for OpenEdge (`Progress Dev AS for OE` in the licensing) — 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) — configured as a secure web server for OpenEdge application deployment.

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

A production instance of PAS for OpenEdge is not meant to “just work” out of the box. It 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 staging your instance using a production license. The staging environment is a test sandbox that replicates the production servers and environment.

For details, see the following topics:

- Development versus production instances
- Missing and disabled features in production
- Licensing and installation
- PAS for OpenEdge configuration tools
- What you need from the developer

- Move to production checklist

Development versus production instances

The difference between a development server and a production server is largely a matter of security configuration.

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

For example, in a development environment anyone might be allowed to shut down the server, whereas, in a production environment, only a system administrator with appropriate privileges would be 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 `oeabl.war` and contains a default ABL application that is name of the instance.
- The Tomcat Manager web application deployed as `manager`. Tomcat Manager provides monitoring and administration tools. Tomcat Manager should only be deployed in 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 instance, there is only an `oeabl`-based ROOT ABL web application that has all transports disabled by default.

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	Secure 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

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

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

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

  The development version 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 version.

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

  Status response is controlled by the `statusEnabled` property in `openedge.properties`. 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 wish to deploy these remote manager applications, see Deploy remote administration applications.

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 on Installation, see the Install OpenEdge book, or see 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 the Licensing topic in Install OpenEdge book.

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.
- **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 allow 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 to explore management REST APIs.
- **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 `oemanager.war` can do, but without opening access to the instance via the web server. For more information, see Use OEJMX to manage and monitor an instance.

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

Utilities for managing keys and digital certificates

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

- `certutil` — Provides all the functions necessary to install and manage root certificates from any Certification Authority (CA) as entries in the root certificate store of an OpenEdge client machine (located in *OpenEdge-Install-Dir*`\certs`).
- `gendomreg` — Creates an encrypted domain registry file that can be used to seal a client-principal token. It is most often used when the token must be validated against multiple domains.
- `genpassword` — Accepts the clear-text value of a password and generates the encoded and encrypted form for the specified password.

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

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

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

.

What you need from the developer

You should configure the staging environment identically to the production environment, then install OpenEdge.

Developers working with PAS for OpenEdge development licenses can provide whole zipped 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 it the instance was running 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 live production, the system administrator has a better understanding of the runtime performance and configuration.

Move to production checklist

The following topics describe configuration and other issues that you should know 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 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	See the Secure an Instance chapter for additional security considerations.
Set environment variables	See the environment variables topic if you want to set the same environment variables each time an instance is launched.

3

Create and configure a PAS for OpenEdge instance

This chapter covers the tasks a system administrator performs to create and configure PAS for OpenEdge instances in production mode. There are different ways to create and manage server instances. This chapter focuses on 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
- Configure a PAS for OpenEdge instance
- Connect to an OpenEdge database
- Register a PAS for OpenEdge instance as a Windows service
- 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, there is no default instance created when you install a production version of PAS for OpenEdge. A default instance implies that ports, name spaces, and other values would be well-known and would result in a security risk. Therefore, before you can 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 related files and logs are stored. The following properties are set when creating the new instance 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, no default instance is provided automatically and you must create one.

To use the `PASMAN` command-line utility to create a new PAS for OpenEdge instance, perform the following steps:

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
```

   - The only required parameters on UNIX 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 the TCMAN Reference and the PASMAN Reference.

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 the OpenEdge Management guide.

To create a PAS for OpenEdge instance using OpenEdge Management, perform the following steps:

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 square 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 or production) the instance is running, perform the following steps:

1. Run the proenv utility.
2. Perform one of the following tasks:
   - Get the environment details by running a `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.

Start and stop a PAS for OpenEdge instance

After you create a PAS for OpenEdge production instance and before you can deploy applications to the instance, you need to 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, perform the following steps:

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 has been reached and the startup process is stopped.
3. Run the `pasman.[bat|sh] env` command to retrieve system information such as 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, perform the following steps:

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. Click Start or Stop.

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`—Use this utility to view or modify configuration properties.
- `pasman feature`—Use this utility to enable or disable server features.
- OpenEdge Management—Use this utility to edit the configuration files using a graphical 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, perform the following steps:

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
   ```

   The command lists all the properties of your instance.
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
   ```

   The command returns the current value of the HTTP port for the specified instance.
4. Update the value of the property using the following syntax:

   ```
   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 has 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, use the following syntax:

   ```
   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, use the following syntax:

   ```
   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, you can edit the configuration properties of an PAS for OpenEdge instance using OpenEdge Management.

To edit PAS for OpenEdge properties using OpenEdge Management, perform the following steps:

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 — Use this tab to configure the instance working directory, agent startup parameter and PROPATH settings, as well as other agent and session procedures.
   - Log Settings — Use this tab to configure logging settings for the agent.
   - Performance Tuning and Limits — Use this tab to specify number of agents, sessions, and timeouts. For more information, see Tune PAS for OpenEdge instances.
   - Network Environment — Use this tab to enable SSL and other network settings.
3. Click Save to save your changes.

Connect to an OpenEdge database

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 will show 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 and to no other session. No single ABL session user connection binds in any way the user connection of any other ABL session.

Two type values specific to PAS for OpenEdge are 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`.

See also:

- Use command-line utilities to connect to an OpenEdge database
- Use OpenEdge Management to connect to the database

For information about database management, see the Database Essentials book.

Use command-line utilities to connect to an OpenEdge database

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

1. Run the proenv utility.
2. Navigate to your OpenEdge work directory:

   ```
   cd install-path/OpenEdge/WRK
   ```

3. Create a folder to store your database:

   ```
   mkdir database
   ```

4. Run the following command in proenv:

   ```
   $DLC/bin/procopy $DLC/sports2020
       openedge-install-path/WRK/database/sports2020
   ```

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

5. Change directory to *openedge-install-path*`/WRK/database`.
6. Start the database server by running the following command:

   ```
   proserve sports2020
   ```

7. Change directory to *openedge-install-path*`/WRK/`*instance-name*`/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 database `sports2000`:

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

   This command tells the instance to connect to database `sports2020` on agent startup.

9. Save the following code to your *openedge-install-path*`/WRK/`*instance-name*`/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:

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

11. Navigate to the 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://localhost:port/apsv -sessionModel Session-Managed".

ret = appHandle:CONNECT(cnctParam).

IF NOT ret THEN
        DO:
           MESSAGE "Server not connected".
           RETURN.
        END.
ELSE
        DO:
          MESSAGE "Connect to PASOE!".
          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 host name 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 wish to use this procedure as a reference.

Use OpenEdge Management to connect to the database

To configure a database connection for a PAS for OpenEdge instance using OpenEdge Management, perform the following steps:

1. Navigate to ABL Application Configuration: Resources → Progress Application Server → *Instance-Name* → ABL Applications → Configuration
2. Specify the connection parameters in the Agent Startup Parameter field under the Startup Parameters and Environment tab. For example:

   `Agent startup parameter: -db` *`path-to-database`*

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, 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, perform the following steps:

1. Run the proenv utility.
2. Run the `pasman service` command. Specify the instance name and the `register` parameter, as in the following example:

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

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

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

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

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, perform the following steps:

1. Run the proenv utility.
2. Stop the target instance if it is running.
3. Issue the `pasman.[sh|bat] delete` command along with the specified filepath and instance name as shown in the following example:

   ```
   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.

4

Application deployment 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 live production system.

Use the staging instance to:

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

After testing and tuning are complete, zip the files to include both r-code and deployed artifacts that can be unzipped and registered on a live production system.

For details, see the following topics:

- Configure properties for deployment
- Stage r-code
- Deployment artifacts
- Package an instance for production
- Update PROPATH in a production instance with zero downtime

Configure properties for deployment

After you create a staging environment, you need to 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. Navigate 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 *instance-path*`/`*instance-name*`/openedge` and *instance-path*`/`*instance-name*`/webapps/`*ablwebapp-name*`/WEB-INF/openedge` to the PROPATH.

Note:

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

Set database connections

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

Perform the following steps:

1. Run the proenv utility.
2. Navigate 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 transports

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

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

1. Run the proenv utility.
2. Navigate 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 other transports should be set to `0`. For example:

```
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 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, perform the following steps:

1. Run the proenv utility.
2. Navigate 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. For example::

```
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. See the Tune PAS for OpenEdge instances chapter for information about tuning a production environment.

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 the Tune PAS for OpenEdge instances chapter.

Modify 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` are run.

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

Stage 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, 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 zipping a complete instance and registering it on the production server.

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 zip the entire instance to easily share it with others. For some applications this may be all that is required. As your services evolve, you may choose to create your own web applications which 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 wish to replace ROOT with another web application, undeploy ROOT and deploy the web application that does what you require when a client hits your server with no path specified. This can be a login page, a redirect, or something else depending on what you require.

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

This topic describes how to:

- Deploy services
- Deploy web applications

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.

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. Since 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`), which contains the configuration information for the application in a standard format.

When deploying ABL web applications, you can pre-deploy and configure transports. This chapter 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. Using the optional embedding of `.r` modules in the web application, 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 ABL services
- Deploy REST services
- Deploy SOAP services
- Deploy WebHandler services
- Deploy an ABL web application
- Deploy remote administration applications

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, perform the following steps:

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

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, perform the following steps:

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

```
deployREST.[bat|sh] path_to_rest/RESTservice.paar ABLwebapp
```

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. Obtain the `.wsm` file and any ABL r-code that supports the API for the service interface from the developer.
2. Move r-code to the PROPATH.

3. Navigate 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 WebHandler services

When you deploy a WebHandler service to a PAS for OpenEdge production instance, the WEB transport must be enabled.

When deploying a new WebHandler service, 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 WebHandler service to an instance:

1. Obtain the r-code that supports the WebHandler service, as well as 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 navigate to the target instance `bin` directory.
4. Add the `handler` properties to the `openedge.properties` using the `oeprop` utility. For example:

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

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. Obtain the `.war` file and any ABL r-code that support ABL web application from the developer.
2. Move any r-code to the PROPATH. *instance-name*`/openedge` directory.
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. Test that the ABL web application is deployed by calling it in a web browser:

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

   where:

   - *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, using the following PASMAN syntax:

```
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 the Configure the instance for production topic.

Deploy remote administration applications

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

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

- Log into 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 manager applications on a production instance.

Run the following command in `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 the following 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
```

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 unzip it to the production instance.

To register a copy of a PAS for OpenEdge instance on a live production machine, perform the following steps:

1. After testing and tuning an instance on a staging environment, zip the instance.
2. Copy and unzip 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 to refer to the new production server.

Update PROPATH in a production instance with zero downtime

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

This feature allows system administrators to:

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

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

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

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

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

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

   This property turns on a process that checks the `openedge.properties` file for changes and interactively passes those changes to the PAS for OpenEdge multi-session agents.

4. Manually restart your PAS for OpenEdge instance to have this change take effect.

5

Secure a PAS for OpenEdge Instance

This topic provides information about security configurations for PAS for OpenEdge instances.

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

The following is a list of security factors that you should consider before running the production version of PAS for OpenEdge:

- SSL/TLS certificate: add a valid key/certificate to your PAS for OpenEdge instance. Do not rely on the private key/ digital certificate sent by OpenEdge for testing in production.
- 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 tomcat username and tomcat password.
- File system access as described in Manage users and remote access.
- Use remote access filters to control client IP addresses.
- Remote administration and monitoring as described in Secure online deployment of a new ABL application.
- ABL application client login model and user account source as described in Enable ABL application authentication.
- Turn off HTTP.

For details, see the following topics:

- Production instance security customizations

- Enable ABL application authentication
- Disable HTTP in production instances
- Use SSL/TLS in PAS for OpenEdge
- Configure Spring Security
- Manage users and remote access

Production instance security customizations

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

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

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

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

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

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

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.

Before running and deploying business applications to a production instance, you should increase the security level by enabling authentication in a staging environment. 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 might require.

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, perform the following steps:

1. Open the `oeablSecurity.properties` file at the security layer you require:
   - *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 `spring.login.model`. The following login models are supported:

Login Model	Description
`anonymous`	The default. All clients have public access. No direct logins or Single sign-on (SSO) 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 via Tomcat realm services and authorize URL access via Spring Security.
`sso`	OpenEdge SSO using ClientPrincipal access tokens. No direct login – only SSO headers.
`oeauth`	OpenEdge support for validating OAuth2 JWT tokens for URL authorization.
`saml`	Users authenticate and authorize using SAML token.

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

User Account Source	Description
`local`	Source is web application `WEB-INF/users.properties` with clear-text passwords. For more information, see Local and Extended Local user accounts.
`extlocal`	Source is the application `WEB-INF/users.properties` with encrypted passwords. For more information, see Local and Extended Local user accounts.
`ldap`	Source is simple LDAP (or Active Directory) server configuration. For more information, see LDAP user accounts.

User Account Source	Description
`oerealm`	Source is a bridge to ABL application maintained user accounts. For more information, see OERealm user account configurations.
`ad`	Source is a simple (constrained) Active Directory configuration. For more information, see Active Directory user accounts.
`sts`	Source is the OpenEdge Authentication Gateway client. For more information see the OpenEdge Authentication Gateway Guide.

4. Update the default `ABLDomainRegistry.keystore` to include your domain name and domain access-code. Perform the following steps:

   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 one.

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

   e. Design the role name pattern that will be used to designate an OpenEdge domain. For more information, see Obtaining a domain name from a role name.

   f. When you have established a role name pattern, construct the Java RegEx pattern that will distinguish the domain name from any other normal user account granted roles.

   g. Set the OEClientPrincipalFilter bean's domainRoleFilter property to the value of the Java RegEx match pattern.

   h. If required by your security practice, set the `OEClientPrincipal.validateClientDomain` property in the `oeablSecurity.properties` to `true`. The `OEClientPrincipal.validateClientDomain` property is a Boolean value that when set to true and when there is a domainRoleFilter defined, it compares the domain specified by the client to the domain produced by the domainRoleFilter. 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 will also need to modify the `oeablSecurity*.xml` file to access your authentication service.

Disable HTTP in production instances

As a security practice, you should disable HTTP in a production instance. To achieve this, perform the following steps:

1. Run the proenv utility.
2. Disable the HTTP feature for an instance with the PASMAN utility. For example:

   ```
   pasman.[sh|bat] feature HTTP=off -I instance-name
   ```

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

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

Use SSL/TLS in PAS for OpenEdge

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

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

Note:

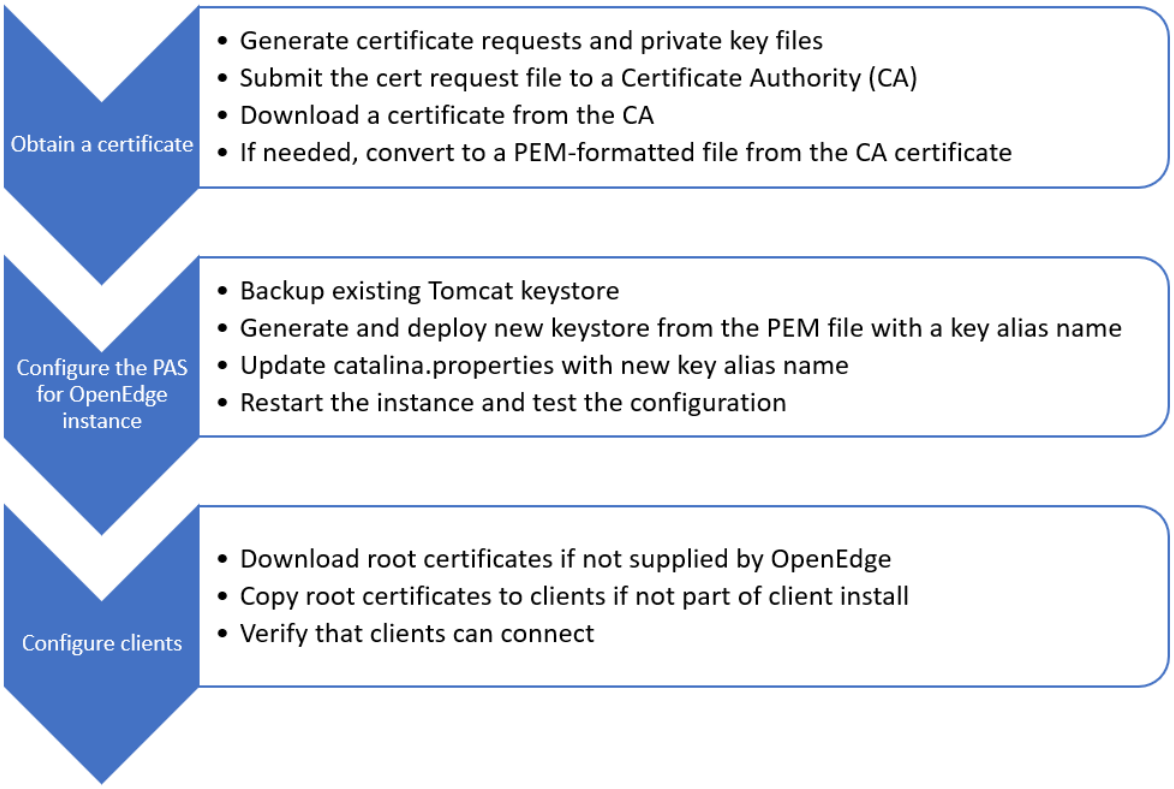
TLS evolved from and superseded SSL. TLS is considered to be more efficient and secure than the older SSL. Even though TLS and SSL are separate and distinct protocols, the PAS for OpenEdge configuration process is the same for both. Therefore we use the term *SSL/TLS*.

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

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

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

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



Important:

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

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

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

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

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

Obtain a certificate from a Certificate Authority

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

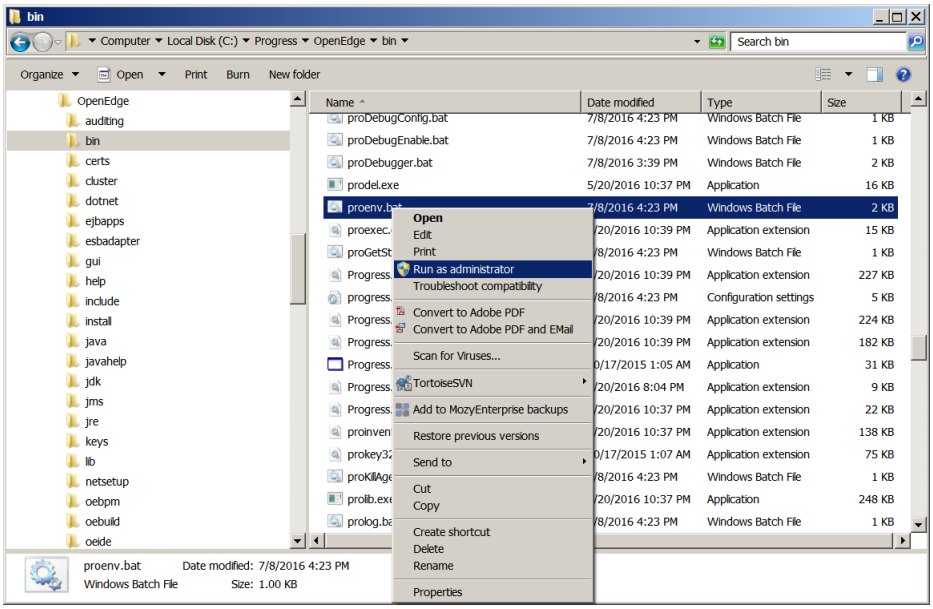
1. Start PROENV and change the directory to *`OpenEdge-Install-Dir`*`\keys\requests`.

For example:

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

Note:

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



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

   For example:

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

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

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

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

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

Important: Make a note of the PEM pass phrase. It will be required when you import the certificate returned from a CA to create a keystore in Step 5 on page 61.

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

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

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

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

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

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

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

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

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

      For example:

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

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

      For example:

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

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

   c) When prompted enter the password you used when you created the keystore (i.e. the `.pk1` file) in Step 2 on page 60.

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

Configure a PAS for OpenEdge instance for SSL/TLS

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

Note: You will need the password that you used when you ran PKIUTIL to create a public key certificate request file (`MyCert.pk10` in the example procedure described in Obtain a certificate from a Certificate Authority on page 59).

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

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

   For example:

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

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

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

   For example:

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

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

3. When prompted enter the password you used when you created the keystore (i.e. the `.pk1` file). See Obtain a certificate from a Certificate Authority on page 59.
4. Use the `sslc` command to verify what is now in the Tomcat keystore.

   For example:

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

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

For example:

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

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

6. Restart the instance and test.

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

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

Configure clients for SSL/TLS

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

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

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

   This directory should be where you copied the SSL/TLS certificates as described in Obtain a certificate from a Certificate Authority on page 59.

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

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

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

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

   Note:

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

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

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

5. Test the client.

Configure Spring Security

PAS or OpenEdge employs the Spring Security framework to provide enterprise-level application security with regard to:

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

Spring Security is a Java-based framework provided in the `$DLC/common/lib` directory of the OpenEdge install.

Implement a Spring Security model by supplying a reference to its configuration template in a `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

Single sign-on (SSO) support

Single sign-on (SSO) in OpenEdge refers to the creation of a sealed security token called the `ClientPrincipal` that represents a user identity within a security domain. The `ClientPrincipal` is created after Spring Security successfully completes authentication and authorization. The `ClientPrincipal` retains the user credentials and remains available within a web application, eliminating the need for a user to login again.

In PAS for OpenEdge, basic Spring Security token generation is extended to create an OpenEdge `ClientPrincipal` object. Creation of the `ClientPrincipal` is enabled by default by this property in the `oeablSecurity.properties` file of the web application:

```
OEClientPrincipalFilter.enabled=true
```

Set the property to `false` to disable the `ClientPrincipal`. You can set other `ClientPrincipal` properties in the `OEClientPrincipalFilter` bean section of the `oeablSecurity.properties` file. In particular, you will want to set the `OEClientPrincipalFilter.domain` property to specify where the `ClientPrincipal` is valid.

After domain and other properties are set for the `ClientPrincipal`, you can configure SSO properties in the `OpenEdge SSO support` section of the `oeablSecurity.properties` file.

See the `oeablSecurity.properties.README` file for more information on the properties and values for the `ClientPrincipal` and for SSO.

Note: In PAS for OpenEdge, SSO is available for client access via the APSV and REST transports but not for the SOAP transport.

CORS support

Cross-origin resource sharing (CORS) is a W3C group standard that allows an HTTP client to access a web page from one domain in order to access resources located in another domain. Such "cross-domain" requests are otherwise forbidden by a default, same-origin security policy of a web browser. The CORS standard defines a way in which a client can ask a web server if it can access the cross-origin request. The configuration of the web server determines if the cross-domain request will be granted.

The CORS standard was designed primarily to support JavaScript clients, however it may be applicable to other clients as well.

The CORS standard is implemented through HTTP headers that allow resource access to permitted domains. Most modern browsers support these headers and enforce the restrictions these headers establish. Additionally, for HTTP request methods that might affect user data (in particular, methods other than `GET` or `POST` with certain MIME types), the specification mandates that the browser preflight the request. Preflighting is a check to determine if the HTTP request is safe to send to the other domain. The server solicits supported methods from the other domain with an `HTTP OPTIONS` request header. Upon approval of the preflight request, the actual HTTP request is sent. Servers can also notify clients whether user credentials (including cookies and HTTP authentication data) can be sent with requests.

Note:

Although CORS support is extended to modern browsers, check to ensure that your browser supports the CORS standard.

You must manage security constraints (for example, roles) in the `oeablSecurity.properties` file of the web application. See the *OECORSFilter bean* of the `oeablSecurity.properties.README` for more information.

A CORS enabled server or web application classifies all HTTP requests as:

- A CORS request that contains the HTTP `Origin` header
- A **preflight** request that contains the `Access-Control-Request-Method` header in an `OPTIONS` request
- A **generic** request that does not contain any CORS HTTP headers

OpenEdge uses a third-party Java open source package, `CORS Filter`, in which all of the primary CORS functionality resides. OpenEdge has integrated CORS filter into the Java container web applications by implementing a Spring Security filter bean so that the CORS filter can be configured from within the Spring Security configuration files, with all of the other web application security.

Therefore, the CORS support is already incorporated into the web application security policy, and is enabled for all generic requests and CORS requests from any domain.

For more information on CORS standard and the advances in the standard, see the documentation at http://www.w3.org/TR/cors/.

Note: CORS can be used as an alternative to the JSON with padding (JSONP) pattern. While JSONP supports only the `GET` request method, CORS also supports other types of HTTP requests. Using CORS enables a web programmer to use the XMLHttpRequest API, which supports better error handling than JSONP. Also, while JSONP can cause XSS issues where the external site is compromised, CORS allows web sites to manually parse responses to ensure security.

Use CORS

To add CORS support with a PAS for OpenEdge instance, perform the following steps

1. Open the `WEB-INF/oeablSecurity.properties` file of your web application.
2. Find the `OECORSFilter` bean.
3. Set `OECORSFilter` properties.

   See the `WEB-INF/oeablSecurity.properties.README` for information about OECORSFilter properties.
4. After saving your changes, restart the PAS for OpenEdge instance.

Manage users and remote access

PAS for OpenEdge can support one or more Tomcat realm definitions. A *realm* is a collection of usernames and passwords, and it includes the roles associated with each of those users. A *role* controls the amount of access a given group of users has. All authorizations in PAS for OpenEdge are role-based. Access privileges cannot be granted on a user-by-user basis.

PAS for OpenEdge retains the roles defined in the default Tomcat MemoryRealm, but also adds a set of PAS for OpenEdge roles that map to the Tomcat roles and allows for consistency across the product. For example, `ROLE_PSCadmin` allows unrestricted administrator access, and it maps to `admin-gui`, `admin-script`, `manager-gui`, `manager-script`, and `manager-status` roles in Tomcat. These roles are described in Managing users and roles for Manager and Host Manager applications.

Remote access can be handled by using remote access filters, which is described in Secure online deployment of a new ABL application and Remote access filters.

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 will have access to the files on the new instance.

On Windows, no user or group access control lists (ACLs) are set during installation. You should manually set permissions on the `%DLC%\servers\pasoe` directories, sub-directories and files to prevent unrestricted access.

Role-based user authentication

For user authentication, the PAS for OpenEdge comes configured for use with Tomcat MemoryRealm. The MemoryRealm accesses user authentication information stored in *instance-name*`/conf/tomcat-users.xml`. You should replace the MemoryRealm with a different type of realm such as JDBC or LDAP in a production instance. You can have multiple realms of different types configured at the same time.

PAS for OpenEdge retains the roles defined in the default Tomcat MemoryRealm, but PAS for OpenEdge also includes a new set of roles that include the Tomcat roles. The following table lists the new roles and what existing Tomcat roles they map to, if any:

Table 6: PAS for OpenEdge roles mapped to Tomcat roles

PAS for OpenEdge role	Includes Tomcat Manager and Host Manager roles	Description
ROLE_PSCAdmin	admin-gui admin-script manager-gui manager-script manager-jmx manager-status	Unrestricted administration access. • The Tomcat admin roles allow access to the HTML GUI and text interfaces of the Host Manager application • The manager roles allow access to the HTML GUI, text interface, JMX proxy, and status pages of the Manager application.
ROLE_PSCOper	admin-script manager-script manager-status	Restricted operator administration access. The Tomcat roles allow access to the text interfaces of the Host Manager and Manager applications as well as the status pages in the Manager application.
ROLE_PSCUser	none	Non-priviliged access
ROLE_PSCNone	none	No access, for user account authorization testing

Note: If you use a realm(s) other than the default MemoryRealm, that realm must have the roles listed in the table above.

PAS for OpenEdge comes with three default test accounts configured in *instance-name*/conf/tomcat-users.xml, as described in the following table:

Table 7: PAS for OpenEdge default user accounts

Username	Password	Role(s)
tomcat	tomcat	ROLE_PSCAdmin, ROLE_PSCOper, ROLE_PSCUser **Note:** Replace the ROLE_PSCUser in production.

Username	Password	Role(s)
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.

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. Secure ABL application deployment to a production instance requires a securely-deployed Tomcat Manager, as well as other security considerations, that are detailed in this topic.

Note: The best practice is to not deploy applications to an online instance. However, there may be circumstances where this is required. If you choose to enable this capability, ensure that you appropriately secure your instance.

Deploying Tomcat Manager

Online deployment of ABL applications requires functionality provided by the Tomcat Manager. The Tomcat Manager is a web application that provides administrative access for configuring instances and deploying applications. For security reasons, the Tomcat Manager is not deployed by default on production servers.

To deploy Tomcat Manager, use the following command:

```
proenv> pasman deploy -I instance-name $DLC/extras/manager.war
```

Securing 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. Therefore, when you enable Tomcat Manager to allow online deployment of ABL applications, you should limit access to the Tomcat Manager to only 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 a request against one or more *regular expressions*, and 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, its format is x:x:x:x:x:x:x:x. That is, the IP address for 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.

Other Security Considerations

Use of the Tomcat Manager or other management applications require careful security configuration. Since the Tomcat Manager application allows for remote deployment of web applications, it can be targeted by attackers. This is often due to weak passwords and publicly accessible Tomcat instances with the Manager application enabled. If the Manager application is enabled, follow these security guidelines:

- Ensure that any users permitted to access the Tomcat Manager application have strong passwords.
- Change the default tomcat user with tomcat password.
- Use roles to limit deploy capabilities to authorized users.
- Use the 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 the PAS for OpenEdge security configuration files.

Deploying an ABL Application

Once the Tomcat Manager is enabled and secured, you can use the PASMAN command to deploy an ABL application. When deploying a web application that requires tailoring to a running PAS for OpenEdge instance, a restart is normally required to load that web applications context. For online deployment, you can use the -l (lowercase L) option to load the context of the application on a running server without requiring a server restart.

Here is an example of deploying an oeabl.war based web application:

```
proenv> pasman deploy -I instance-name -l -u myuid:mypwd -v $DLC/extras/oeabl.war
ABLapp-name
```

Note: The Tomcat Manager (manager.war) must already be deployed for the -l switch to work. If the Tomcat Manager is not deployed, you are prompted to restart the instance. For production instances, you have to deploy the Tomcat Manager first before you can use this switch. Also note that the -l switch can be used only in conjunction with the -u parameter.

Remote access filters

The PAS for OpenEdge is configured with filters that allow you to deny access based on IP addresses or host names. These filters are implemented as valves in the `/conf/server.xml` file of the instance. The following snippet from the default `server.xml` shows that the filters are set to allow access by anyone:

```
<Valve className="org.apache.catalina.valves.RemoteHostValve"
 allow=".*"/>
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
 allow=".*"/>
```

The RemoteHostValve and RemoteAddrValve filters can take two attributes:

`allow`

A comma-delimited list of regular expressions that a client must match in order to be allowed access.

`deny`

A comma-delimited list of regular expressions that, when matched, excludes a client from access.

Note: Enabling remote hostname filters requires DNS hostname resolution, which can result in slower performance.

6

Tune PAS for OpenEdge instances

A well-tuned PAS for OpenEdge instance balances the use of operation system (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 will degrade client response times or cause server outages.

This topic 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

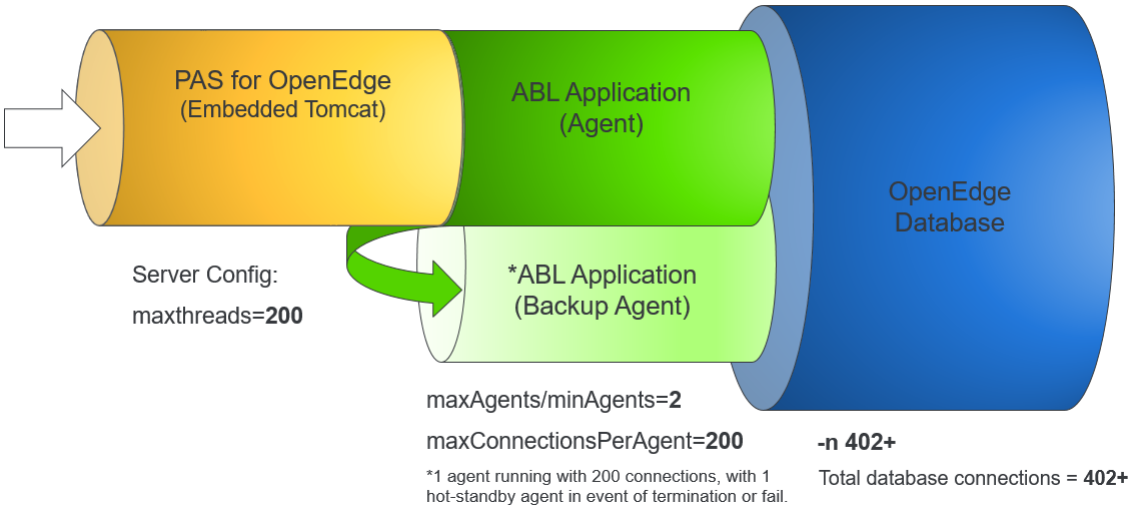
For details, see the following topics:

- 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 MS-agents

PAS for OpenEdge tuning recommendations

Before you read a detailed account of PAS for OpenEdge tuning, 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, they could improve the performance of your application in production.

The following illustration 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 have been tuned:



The illustration shows a PAS for OpenEdge instance that:

- Can run 200 concurrent client requests to a single stateless/statefree 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` utility. For more information, see Use PASMAN to configure an instance.

The sum total of the above 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 high-water 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 sum 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 in the behavior of 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 on 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`	Start two MS-agent OS processes: one handles all client requests and one remains idle in hot-standby mode.
• `minAgents=2` • `maxAgents=2`	Maintain two running MS-agent processes at all times, where if one agent stops a second is automatically started.
`maxConnectionsPerAgent=200`	Set to the maximum number of concurrent client requests handed by the PAS for OpenEdge server (Set equal to `psc.as.executor.maxthreads`).
`maxABLSessionsPerAgent=200`	Set equal to the `maxConnectionsPerAgent` in stateless/statefree 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
```

The described configuration runs the maximum client request load (`maxConnectionsPerAgent`) of 200 using one agent OS process and the second agent OS process stays in standby mode until the first agent OS process exits by system administrator directive or failure. The system administrator does 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 OS process, CPU, memory, or IO limits. For more information about tuning ABL application configurations, see Tune OpenEdge web applications and MS-agents.

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 instance(s).
2. Tuning the OpenEdge application and any third-party web applications.
3. Tuning the OpenEdge multi-session agent (MS-agent).

The following is a general set of steps to get started with tuning 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 as needed until all web applications start cleanly.
2. Determine and set the maximum number of ABL sessions that a single MS-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 MS-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 MS-agents as `(`*`session-pool-size / maximum sessions per MS-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 MS-agent can have. The goal is to maximize the number of concurrently running client requests that one MS-agent can process. The hardware CPU resources and the level of IO switching of the individually executing ABL requests influence the value. This number needs to 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 MS-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 MS-agent starts when it is created. This number should be kept to a minimum because it affects how long it takes for the PAS for OpenEdge instance to become available to handle client load. The goal is to initialize enough ABL sessions to handle the initial client load, and then to gradually scale up (trading off initial client response times) until the PAS for OpenEdge instance 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 still not met the performance requirements, you need to 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 involves 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
- The number of HTTP client connections
- The size of the thread pool used to execute HTTP client requests
- Client connection timeout (elapsed time without an HTTP request)
- The queue size for parking 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 exist 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 exist 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 the 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 can be deployed on the PAS for OpenEdge instance and the number of clients 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 needs to 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 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 adding 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 8KB, so the maximum psc.as.msg.maxpostsize is not an issue. If you are using REST or SOAP clients with very large ProDataSet transfers, this property setting may become important.

The secondary goal is to modify the HTTP message-handling if the default settings do not allow 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 a HTTP or HTTPS message.
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.
psc.as.executor.minsparethreads	10	Minimum number of threads retained in the pool to service client requests.

Property	Default	Description
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 detracts from the amount of JVM memory available to the applications. Having too few threads detracts from the ability of the instance to handle concurrent client requests.
- Another factor in setting the thread pool properties is 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 slack 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` 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 which 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 may 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 just 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, 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 may 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 MS-agents

After you tune PAS for OpenEdge client requests, you can begin tuning the OpenEdge web applications and MS-agent processes to achieve the following goals:

- Control the size of the ABL session pool in the Session Manager
- Manage resource consumption within an MS-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 strike a balance between PAS for OpenEdge instance startup times, supporting client requests to your ABL business logic, and the response time to those clients.

One of the realities in getting the most performance from your PAS for OpenEdge instance is that the OpenEdge web applications and the MS-agent process execution of client requests is gated by your tuning 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, the CPUs, and the 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 MS-agents, and the local network socket connections.

The file can be edited managed using the *instance-name*`/bin/oeprop.[bat|sh]` command line utility, OpenEdge Management, or a text editor.

This topic describes the `openedge.properties` and how they can be used to tune the MS-agent and ABL sessions of a PAS for OpenEdge instance:

Property [AppServer.SessMgr]	Default	Description (per ABL Application)
`numInitialAgents`	`1`	Number of MS-agent processes to create at server startup.
`maxAgents`	`2`	Maximum number of MS-agent processes that can exist.

Property [AppServer.SessMgr]	Default	Description (per ABL Application)
maxABLSessionsPerAgent	200	Maximum number of ABL sessions per MS-agent process.
maxConnectionsPerAgent	200	Maximum number of network connections between Session Manager and an MS-agent.
requestWaitTimeout	15000	Maximum time in milliseconds that a client request is queued waiting for an ABL session before an error is generated.
idleSessionTimeout	300000	Maximum time in seconds that an ABL session remains idle before it is shut down.
idleAgentTimeout	1800000	Maximum time in seconds that a MS-agent remains idle before it is shut down.
agentListenerTimeout	300000	Maximum time that the Session Manager waits for an MS-agent to report it has started before an error is generated.
minAgents	1	Minimum number of agents expected at any time. If the number of agents drops below this number, 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, the minAgent value is lowered to match the maxAgents. The Session Manager ensures that there is always one MS-agent available at all times before running the request.
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 has not been accessed more recently than the specified timeout for that property is terminated. If this property is set to 0, idle resource checking is disabled.

Property [AppServer.SessMgr]	Default	Description (per ABL Application)
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 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 MS-agent)
numInitialSessions	2	Number of ABL sessions started at MS-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 MS-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 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 MS-agents, local socket connections, and ABL sessions:

1. The instance receives a client request from an oeabl web application transport.
2. The first MS-agent in the list of MS-agents is set as the current agent.
3. If the current MS-agent has an unused agent connection and a free ABL session, the MS-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, whereby 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 MS-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 MS-agent local agent connections reaches the maximum, and another MS-agent exists in the MS-agent list, the next MS-agent is set as the current agent, and executes the client request.
6. If the maximum number of MS-agents has been reached, you should create a new MS-agent. It is added to the end of the MS-agents list, set as the current agent, and executes the client request .
7. If you receive a large number of client requests, queue the request if the maximum request queue size is not exceeded. Queued requests are de-queued and executed as soon as a MS-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 MS-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 MS-agent. During the startup of a MS-agent, the Session Manager holds until the MS-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 MS-agent either unresponsive or unable to start.

The Session Manager uses the `numInitialAgents` property to begin building a list of MS-agent processes. The MS-agent process uses the `numInitialSessions` property at startup time to begin building the list of physical ABL sessions. For example: 1 MS-agent with 5 initial ABL sessions yields an initial ABL session pool size of 5. Starting 3 MS-agents with 5 initial ABL sessions each yields an initial ABL session pool size of 15.

Tuning tips:

- The longer time the `agentListenerTimeout` property is set, the longer the PAS for OpenEdge instance takes to start if the MS-agent process has startup failures. Too short of a time, and the PAS for OpenEdge instance may start sooner but the Session Manager may consider the MS-agent unavailable.
- Setting the number of MS-agent processes and physical ABL sessions to start at initialization time is dependent on how long the ABL application implementation takes to run its startup procedure. A larger number takes more time, which causes the PAS for OpenEdge instance to not handle incoming client requests for a longer period of time. A smaller number and the first client requests handled by PAS for OpenEdge will have slow response times while new ABL sessions or MS-agent processes, or both, are started.

MS-agent parameters and ABL sessions

After 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 MS-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 MS-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 MS-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 MS-agents as small as possible by maximizing the number of ABL sessions and concurrent client requests for each MS-agent.
- The more MS-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 MS-agent and a few ABL sessions.
- Scaling to handle more MS-agent processes is normal when the ABL application supports stateful client requests. Overhead for stateful clients is low because they are directly dispatched for execution to an exact ABL session within an exact MS-agent, without the need to find an idle ABL session.

As stated previously, the Session Manager goal is to conserve resources. It does this by periodically searching for MS-agent and client sessions that have been idle in excess of the `idleAgentTimeout` and `idleSessionTimeout` periods respectively. When an idle MS-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 valleys, this may lead to thrashing whereby client response times decrease because MS-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 MS-agent and ABL session startups and idle resource shutdowns occur, increase the time they can remain idle before being harvested.
- You also have the option of manually controlling the harvesting of idle MS-agents and ABL sessions. To disable the automatic idle resource harvesting, configure the `idleResourceTimeout` property to 0 and use the JMX or remote OpenEdge administration plugin to manually harvest idle resources.

Manage concurrent request execution

An MS-agent can only 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 MS-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 MS-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 MS-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, you need to increase the number of socket connections and ABL sessions because many of them will have high idle times whereby 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.

7

Load balance PAS for OpenEdge instances

This topic describes the load balancing options for PAS for OpenEdge. Load balancing in PAS for OpenEdge is accomplished through industry standard HTTP options. This topic includes an overview of configuration tasks for each option, and where applicable, describes the pros and cons of the option.

There are many more third-party load balancing products available. The load balancing options covered in this topic have been 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
- 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.

- **Adding and removing agents**— The next step in scaling an instance is to adjust the number of agents if you can't 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

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. Applications deployed to REST, SOAP and WEB transports implicitly use HTTP sessions. By default, the ROOT application for APSV is enabled. For additional APSV transport applications, 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
```

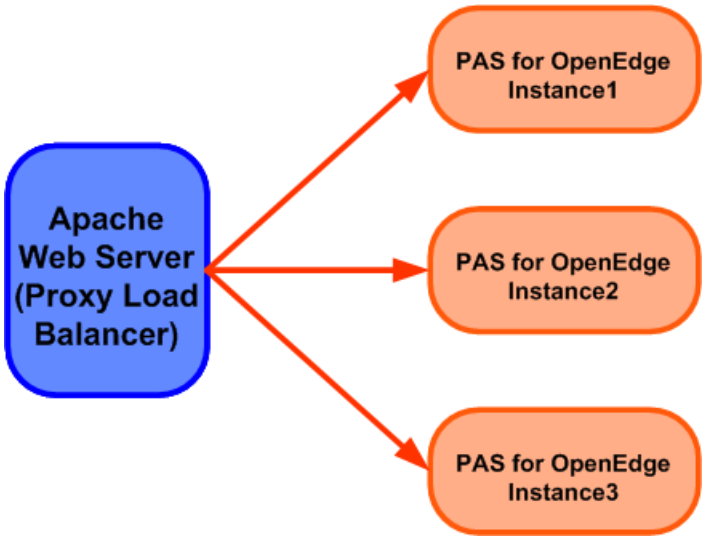
   It is recommended to enable chunking when you enable HTTP session to improve performance.

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 Configure APSV transport properties. If you want to implement load balancing see Load balance PAS for OpenEdge instances on page 87.

Apache proxy host load balancing

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



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

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

---

Note: The disadvantage in this type of load balancing is the Apache server does not monitor the running status of instances. If an instance crashes or is otherwise unavailable, the Apache load balancer continues to try to send requests to the unavailable instance and requests will be lost. Compare with Tomcat load balancing on page 92, where the running status of instances is monitored.

Requirements for Apache proxy load balancing

The requirements for implementing Apache proxy load balancing are:

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

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

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

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

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

Configuring the Apache server for proxy load balancing

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

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

Note: In addition to configuring the Apache server, you must configure each PAS for OpenEdge instance. See Configuring PAS for OpenEdge instances for proxy load balancing on page 91

Loading proxy modules

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

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

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

Referencing instances and configuring session types

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

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

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

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

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

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

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

In the example above notice that:

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

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

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

Configuring PAS for OpenEdge instances for proxy load balancing

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

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

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

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

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

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

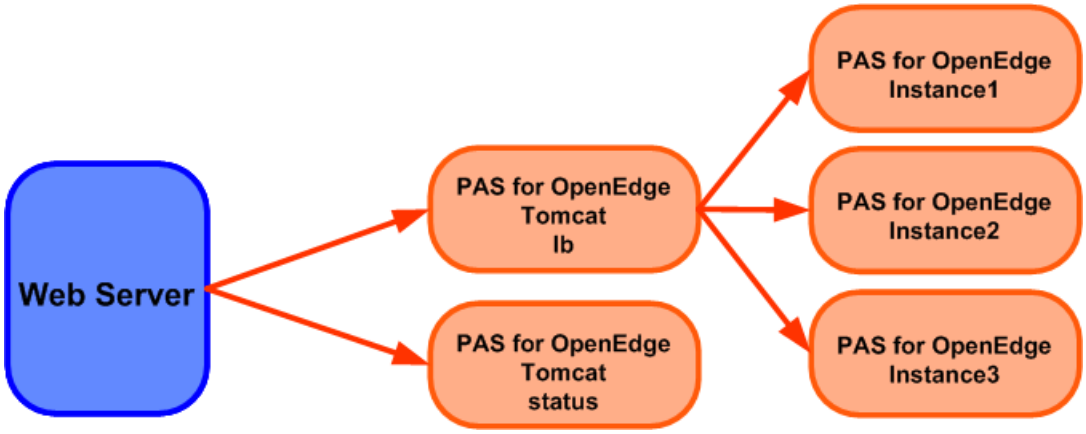
Note: The `jvmRoute` alias translates to the route value in the `http.conf` file of the Apache server. See Configuring the Apache server for proxy load balancing on page 90 for more information.

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

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

Tomcat load balancing

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



Some advantages of this configuration, compared to Apache proxy host load balancing on page 89, are:

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

Requirements for Tomcat load balancing

The following are the requirements to implement Tomcat load balancing:

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

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

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

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

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

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

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

Configuring an Apache web server for Tomcat load balancing

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

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

Perform the following steps:

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

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

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

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

```
#workers.properties load balancing config

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

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

   For example:

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

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

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

Configuring PAS for OpenEdge for Tomcat load balancing

Configuring PAS for OpenEdge to enable Tomcat load balancing involves:

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

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

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

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

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

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

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

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

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

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

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

   Note:

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

Amazon load balancing

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

Configuring Elastic Load Balancing and Auto Scaling

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

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

Table 8: Elastic Load Balancing and Auto Scaling components

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

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

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

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

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

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 on clustering and load balancing orchestration using containers, see the Kubernetes Documentation and the Docker swarm documentation.

For more information on running PAS for OpenEdge in a Docker container, see Get Started with containers and PAS for OpenEdge.

8

Monitor PAS for OpenEdge instances

PAS for OpenEdge offers system administrators numerous tools and avenues for monitoring their instances and applications in production. These monitoring tools include:

- Enhanced MDC logging framework
- Log file viewer from OpenEdge Management
- The OpenEdge Debugger application
- JMX access via the OEJMX utilty
- REST APIs which are documented by the Swagger UI
- The OpenEdge HealthScanner
- Server-side ABL performance profiling

For details, see the following topics:

- Configure logging in PAS for OpenEdge
- Enable Debugging
- Use OEJMX to manage and monitor an instance
- Use Server-Side ABL Performance Profiling
- Use Swagger UI to explore management REST APIs
- Use the OpenEdge HealthScanner
- Use the log file viewer from OpenEdge Management

- Troubleshoot problems with an instance

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 via tokens in log entry formats. The Logback framework allows for consistency across all of OpenEdge and Tomcat.

In OpenEdge, the Logback XML files are localized into properties files which 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, as well as 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 log file patterns
- Configure MDC access for header requests

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 permits control of logging for the Tomcat server and its distributed web applications:

Tomcat subsystem	Log file name	Property name	Value definition
Java 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 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.

Tomcat subsystem	Log file name	Property name	Value definition
		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 may 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 executes 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 File Name	Required	Configuration File	Notes
abl-app-name.*date*.log	yes	*instance-name*/conf/logging-pasoe.properties	Full configurability of output format and MDC content.
abl-app-name.agent.*date*.log	yes	• *instance-name*/conf/logging-pasoe.properties • *instance-name*/bin/openedge_server.[sh|bat]	Restricted ability to configure the output format.
*abl-app-name*_authn.*date*.log	no	*instance-name*/conf/logging-pasoe.properties	Created when the Spring 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 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 file name	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 file name	Property Name	Value Definition
*abl-app-name*_authn.*date*.log	psc.as.logging.authn.fileName	Defines the output file name 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 file name	Property Name	Value Definition
*abl-app-name*_authz.*date*.log	psc.as.logging.authz.fileName	Defines the output file name 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 contents.

Default log file 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.

Default log file name	Web Application	Property Name	Value Definition
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.
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.

ABL agent configuration

The ABL agent log file (*abl-app-name*.agent.*date*.log) has very limited control over the log file output. You can select enhanced or classic logging using the ENHLOGGINGOPTS environment variable contained in *instance-name*/bin/openedge_setenv.[sh|bat]. The default is enhanced logging.

The enhanced logging format allows for compressed output format, client request-Id logging, and ISO date-time formats, while the classic logging format does not.

The value of this ENHLOGGINGOPTS permitted configuration includes:

Value	Behavior
undefined	Removing or commenting out the ENHLOGGINGOPTS=; export ENHLOGGINGOPTS statement results in 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.

The agent log file of an ABL application (*abl-app-name*.agent.*date*.log) contains entries from internal ABL Sessions and threads that do not correspond to actual PAS for OpenEdge client requests. Two special request-Id values are used to denote logging entries that originate from the multi-session agent run-time:

Request-ID	Description
mtapsv:-:?	The ABL Session assigned to the client request is in the initialization process. • 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 one was not passed to it. The request continues to execute but without client request tracking. • The request originator is ? (unknown) • The transport is ? (unknown) • The Client message number is ? (unknown)

Any message or standard output within the ABL agent log is preceded by **R-** when enhanced logging is enabled. If the message only comes from the internal multi-session agent code then the output appears as **R-?:?:?** or simply **R-mtapsv:-:?**. When a message is output during execution of code from a transport request, then OERequestInfo is available and the expected values are provided for web application, transport, and request-Id.

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 require.

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 via 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 *instance-name*/conf/logging-tomcat.properties:

```
# 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 9: Tomcat access log supported MDC fields

MDC field	Description
`%reqAttribute{OEReq.requestId}`	Track 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}`	Request 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.

Note:

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 *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, 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 10: Supported HTTP Request MDC Field Names for Spring Security and Session Manager logs

Logging Field Name	Logging Field Description	Tomcat Access Log Syntax	ABL Application Log Syntax
`OEReq.userId`	The Spring Security authenticated user-id who is identified as the client from the server.	`%{OEReq.userId}r`	`%X{OEReq.userId}`
`OEReq.sessionId`	The HTTP Session ID assigned to the current PAS for OpenEdge client login session (if server login sessions are configured)	`%S`	`%X{OEReq.SessionId}`

Logging Field Name	Logging Field Description	Tomcat Access Log Syntax	ABL Application 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.requestId}`
`OEReq.webapp`	The Java web application context name the PAS for OE client request executes in	n/a	`%X{OEReq.webapp}`
`OEReq.transport`	The {apsv | soap | rest | web | file} PAS for OE service interface transport name the client's request executes in	n/a	`%X{OEReq.transport}`
`OEReq.remoteHost`	This field contains the remoteHost DNS address obtained from the PAS for OE network connection	`%h`	`%X{OEReq.remoteHost}`
`OEReq.remoteAddr`	This field contains the remoteHost IP address obtained from the PAS for OE network connection	`%a`	`%X{OEReq.remoteAddr}`
`OEReq.serverName`	This field contains the name of the PAS for OpenEdge server	`%v`	`%X{OEReq.serverName}`
`OEReq.URI`	This field contains the HTTP request's relative URI within `OEReq.webapp`	n/a	`%X{OEReq.URI}`
`OEReq.protocol`	This field contains the HTTP request scheme (http / https)	`%H`	`%X{OEReq.protocol}`
`OEReq.method`	This field contains the HTTP request's method name	`%m`	`%X{OEReq.method}`
`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{OEReqHead.`**`xxxxxx`**`}`

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 Authentication and Authorization logs

The Spring Authentication and Spring 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 Authentication logging fields

The OpenEdge ABL web application login event log file (*abl-app-name*_authn.*date*.log supports the following MDC fields which are designated by the %X token:

MDC field name	Logging Field Description	ABL Application Log Syntax
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 above.

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 which are designated by the `%X` token:

MDC Field Name	Logging Field Description	ABL Application Log Syntax
`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.

Note: 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 MDC fields can be set using the `OEReqHead.xxxxx` token, where `xxxxx` is the case-insensitive HTTP header name. 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 webapps.
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
```

Enable Debugging

Debugging support for PAS for OpenEdge environments is provided by the oedbg.war application. It is possible to debug code running on PAS for OpenEdge only when running from 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. Enter the following command:

```
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.

3. Debugging remains enabled unless you explicitly disable it. If you did not set the ENABLE_OPENEDGE_DEBUGGER environment variable, 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 did set the ENABLE_OPENEDGE_DEBUGGER environment variable, you must also remove the variable.

To enable debugging for a PAS for OpenEdge development instance, follow these instructions:

1. Shut down PAS for OpenEdge instance if it is running. It must be started using the below steps in order for the debugger to be effective.
2. Open a program you wish to debug in Progress Developer Studio and set a breakpoint in it by double-clicking along the left-edge of the line you wish to begin debugging on. This will show a small dot (.) 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 (descibed in the Debug Configuration Settings documentation), and click Debug to start the instance.
6. Once the instance is finished starting, send a request to the instance. Developer Studio switches to the Debug Perspective when the breakpoint is encountered.

For more information, see Debugging an ABL application deployed on a Progress Application Server for OpenEdge.

Use OEJMX to manage and monitor an instance

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

See also:

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

List available MBeans to locate AgentManager details

A running PAS for OpenEdge instance must be available.

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

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

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

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

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

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

With a list of available beans, you can Write a custom OEJMX query to get agents on page 115 .

Write a custom OEJMX query to get agents

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

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

In List available MBeans to locate AgentManager details on page 114, steps are provided to list the available beans. From that list, the operations and attributes for the AgentManager are identified. To collect information about the agents, create a custom OEJMX query file using `getAgents` for the default `oepas1` instance. To write a custom OEJMX query, complete these steps:

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

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

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

Note: A query must be a single line containing no embedded spaces. A query file can contain multiple queries. The best practice is to use explicit names over tokens to guarantee the intended results. In this example, `oepas1` replaces the *APP_NAME* token. For more information about using tokens, see OEJMX utility on page 125.

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

Next you can Get agent information on page 116.

Get agent information

A running PAS for OpenEdge instance must be available. Review Write a custom OEJMX query to get agents on page 115 to get then available agent.

To get agent information, complete the following steps:

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

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

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

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

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

For more details on the OEJMX utility, see the OEJMX utility on page 125.

Generate a stack trace

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

1. Write a query to `getSessionMetrics`.

Example:

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

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

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

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

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

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

   This collect the getSessionStacks details.

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

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

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


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

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


From more information see the Common OEJMX queries on page 118 .

Common OEJMX queries

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

Get active session information

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

Query:

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

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

Result:

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

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

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

Get a stack trace for a session

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

Query:

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

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

Result:

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


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


Get metrics for a specific ABL application

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

Query:

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

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

Result:

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


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


Get hung client connections for a specific ABL application

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

Query:


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


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

Result:

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


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


Get information about active session on a given agent

When to use:

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

Query:


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


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

Result:

Returns session metrics associated with the agent.


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


Get agent session metrics

When to use:

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

Query:


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


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

Result:

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


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

Get Active threads from Tomcat thread pool

When to use:

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

Query:

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

Note: For more information about using tokens, see OEJMX utility on page 125.

Result:

Returns **10** as activeCount.

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

Get APSV Transport metrics for the given ABLApp and WebApp

When to use:

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

Query:

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

Note: For more information about using tokens, seeOEJMX utility on page 125.

Result:

Returns the APSV Transport Metrics for a given OEABL WebApp.

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

OEJMX utility

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

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

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

Syntax

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

Parameters

`-C`

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

`-Q` *queryfile*

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

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

Attribute query:

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

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

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

- APP_NAMEAGENT_ID
- AGENT_PID
- SESSION_ID
- ABL_SESSION_ID

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

`-R`

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

`-O outputfile`

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

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

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

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 of ABL applications running on a PAS for OpenEdge instance.

This document 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:

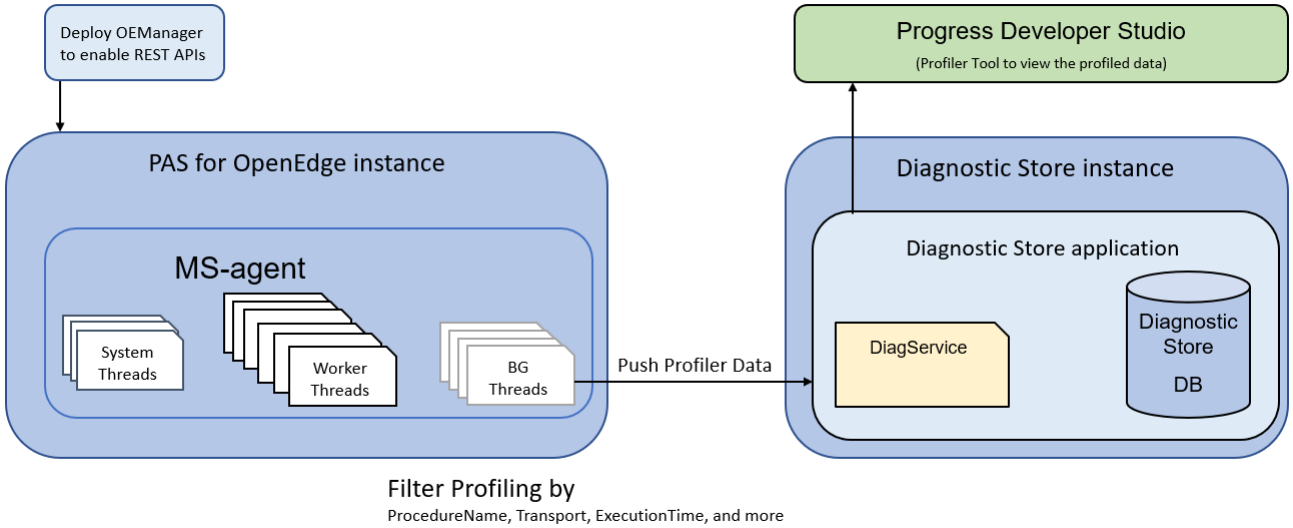
- What is Server-Side Profiling?
- Configure the diagnostic store
- Configure Server-Side Profiling with PAS for OpenEdge
- Supported JSON values for JMX queries

What is Server-Side Profiling?

Use PAS for OpenEdge Server-Side Profiling to gather and monitor run-time performance data of 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 the instance to gather data.

The following diagram illustrates a Server-Side ABL Performance Profiling workflow and architecture:

Figure 1: Server-Side ABL Performance Profiling architectural diagram



When Server-Side Profiling is enabled for a PAS for OpenEdge instance:

1. One or more MS-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 resides 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, perform the following steps:

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. For example:

   ```
   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`. For example:

```
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. For example:

```
[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 by running the following command:
      - 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 the diagnostic store instance starts successfully. For example:

```
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 the Supported JSON values for JMX queries topic.

Before executing the steps in this topic, ensure 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 document.

In this procedure, you are configuring profiling on the target PAS for OpenEdge instance and sending the profiled data to the diagnostic store. Perform the following steps:

1. Ensure the diagnostic store and the target PAS for OpenEdge instance are running. The target instance requires the `oemanager.war` 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`. For example:

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

      where *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\"}"]}
   ```

   where:

   - *host* and *port* to 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 successfully 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"]}
   ```

   where `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, 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. Since this matches the filter and the AdapterMask is REST, this request is profiled and the profiler output sent to the diagnostic store.

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:

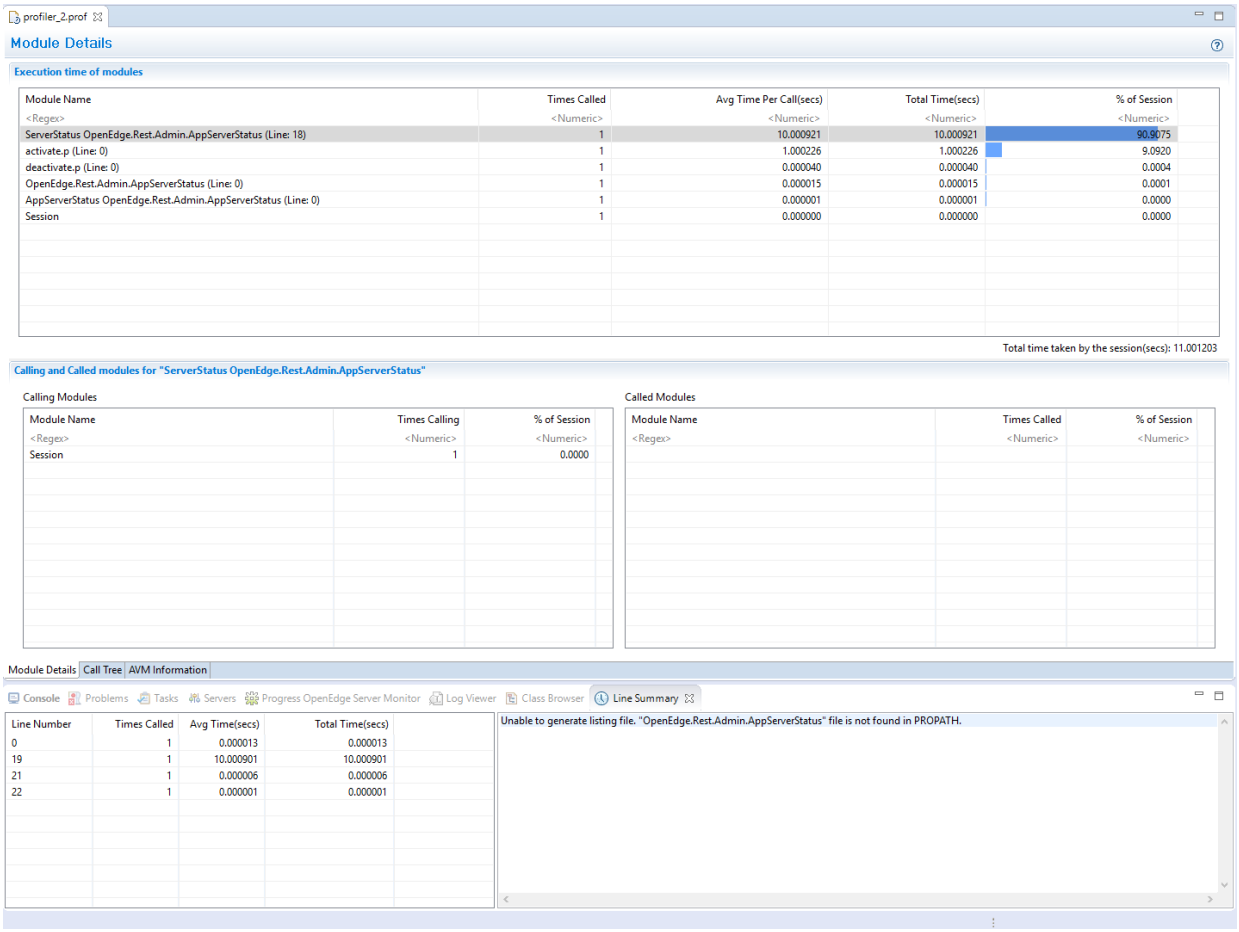
   ```
   DEFINE VAR requestId as Integer NO-UNDO.
   DEFINE VAR fileName as Character NO-UNDO.
   FOR EACH ProfiledRequest:
    requestId = requestId + 1.
    fileName =  "C:/OpenEdge/WRK/profiler_" + STRING(requestId) + ".prof".
       COPY-LOB ProfiledRequest.PerfData TO FILE fileName.
       MESSAGE 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 in Progress Developer Studio, or other 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 2: 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:

- Target URL where the data is to be pushed (required)
- Number of requests to push data for (required):
  - `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 not specified, the settings keep their default values. These settings act as filters for capturing profiled data:

Table 11: 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`	`nil`
`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. Empty string means profile all.	`nil`
`AdapterMask`	String	A CSV filter of adapters. Supported adapters: REST, APSV, SOAP, WEB. `nil` means accept all.	`nil`
`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 much more through this interactive documentation.

Access to Swagger UI is disabled by default to prevent unauthorized users from tampering with an instance.

See also:

- Enable Swagger UI for management REST API access
- Explore and test management REST APIs

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 of `tomcat:tomcat`. Start the instance, at least once to create the oemanager folder structure containing the necessary configuration files.

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, complete the following steps:

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/**` authorizing specific roles to use the interactive documentation.
4. Save the file.
5. Restart the server to apply the changes.

   Use `pasman pasoestart -restart -I` *`instance`* to restart.
6. Enter `http://`*`host`*`:`*`port`*`/oemanager/` to show the management REST API documentation confirming that authorization has been correctly defined. Login credentials are required ensuring 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 133.

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, complete the following steps:

1. Enter `http://host:port/oemanager/` in a browser to access interactive documentation.

   a) 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 **Execute** the API to retrieve a list of available applications.

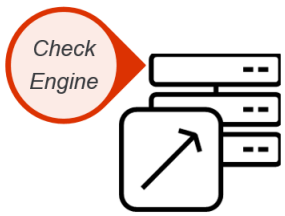
   The response body is JSON formatted and the `applicationName` element provided details about the deployed applications, including which transports are enabled for the application and the version information.
6. Take note of the `applicationName` for use as the request body to find the Agent running the application.
7. Expand the **Agent Manager** section.
8. Click on the `/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 on managing PAS for OpenEdge instances see, Introduction to PAS for OpenEdge

Use the OpenEdge HealthScanner

Like the check engine light for a vehicle, the Progress Application Server for 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. It is not designed to research or resolve the problem, only to flag a server that falls below a healthy threshold.

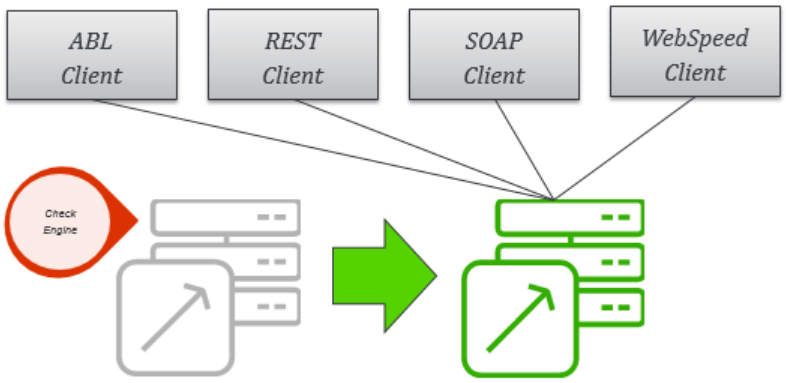


See also:

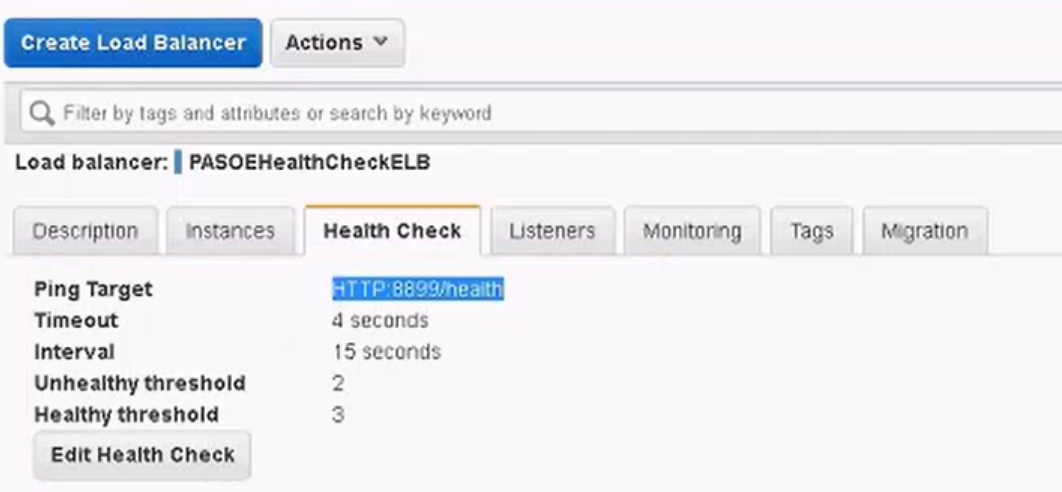
- Primary use case
- How is the data collected?
- Create an instance
- Start the HealthScanner
- Access HealthScanner data
- Display results of a healthy REST call
- Display results of an unhealthy REST call
- Customize HealthScanner reports

Primary use case

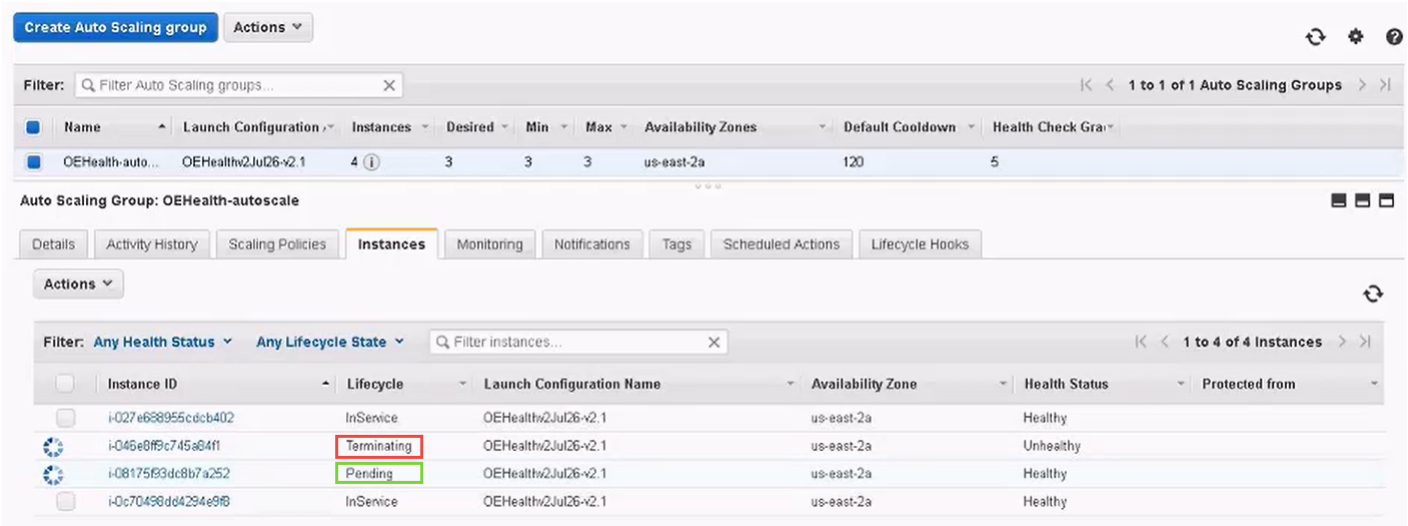
Designed for system administrators, the PASOE HealthScanner checks the overall health of a server assigning a score from 0-100% as well as a status code. System Administrators configure an acceptable threshold for a server to be considered in operational good health. Servers with scores below the threshold can be taken out of service by an elastic load balancer (ELB). The ELB can terminate the server and dynamically replace the failing server before any disruption of service is incurred.



The following is an example of the AWS (Amazon Web Service 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.



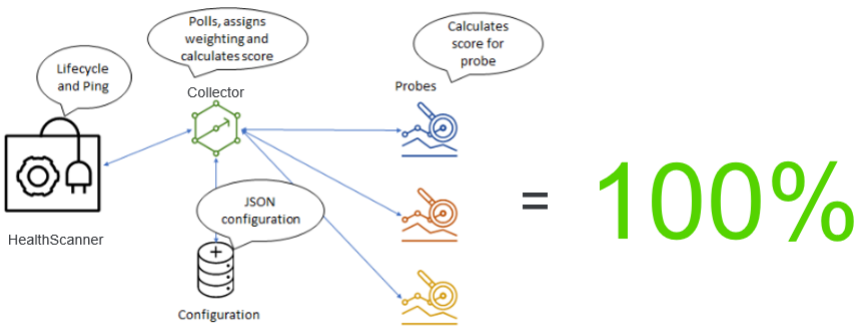
If the ping returns a status code of 500, then the server is automatically terminated, and the new server is brought online providing continuous availability of a healthy server to respond to client requests.



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, collecting 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.



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.

Create an instance

After installing PAS for OpenEdge, use `pasman create` to create an instance. For this demonstration a development instance is used. Additional configuration and security changes are required for a production instance.

1. In Windows, open a `proenv` command prompt.
2. Change to the directory for your new instance, `proenv>cd c:\OpenEdge\wrk`.
3. Enter the following command to create a development instance.

```
pasman create myInstance
```

```
proenv>pasman create myInstance

server instance myInstance created at C:\OpenEdge\WRK\myInstance
done - complete any manual edits at this time.
proenv>
```

Start the HealthScanner

PASOE HealthScanner is intended for use with production instances. When using a production instance, be sure to enable the IP filter to the secure the connection used by the HealthScanner. The following is intended for testing on a development instance.

Enable the HealthScanner

The PASOE HealthScanner is disabled by default, use the `tcman` command to enable.

1. Change to the `bin` directory for the instance. For example:

```
cd instance\bin
```

```
proenv>cd myInstance\bin

proenv>
```

2. List the *feature=value* pairs. Enter the following:

```
tcman feature
```

```
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


proenv>
```

The `HealthCheck=off` shows that the feature is disabled. Download a copy of the tcman Quick Reference guide to learn more about `tcman`.

3. Enable the HealthCheck feature. Use the following:

```
tcman feature HealthCheck=on
```

```
proenv>tcman feature HealthCheck=on


proenv>
```

4. Verify the change, use tcman to poll the HealthCheck value. Enter the following:

```
 tcman feature HealthCheck
```

```
proenv>tcman feature HealthCheck
HealthCheck=on


proenv>
```

Turn on data collection

The data collected by the HealthScanner is based on a series of enabled probes. In this topic, the Health Collector is enabled and the properties controlling the data collection are set in the `appserver.properties` file in the *instance*`\conf` directory.

1. The *instance*`\conf` directory contains an `appserver.properties` file with properties related to enable and to configure the collection of data from the probes.

```
# PAS health collector enable
psc.as.health.enabled=false
# PAS health collector configuration file
psc.as.health.config=health.config
# PAS health collector delay before starting polling
psc.as.health.delay=10
# PAS health collector wait on shutdown for polls to complete
psc.as.health.wait=5
# PAS health collector polling interval
psc.as.health.interval=60
# The timeout for polling each individual probe in milliseconds.
# A value of -1 sets the timeout as # Probes/ Interval
# A value of zero means no timeout.
psc.as.health.timeout=-1
```

2. Enable the collector to collect data from the probes. Enter:

```
tcman config psc.as.health.enabled=true
```

```
proenv>tcman config psc.as.health.enabled=true

proenv>
```

Note: Repeat the property name to display the newly set value. For example:

```
proenv>tcman config psc.as.health.enabled=true psc.as.health.enabled
psc.as.health.enabled=true

proenv>
```

---

3. Restart the server. Enter the following:

```
tcman pasoestart -restart
```

```
proenv>tcman pasoestart -restart
Starting stopped PASOE instance myInstance
........

Start action: start
Initial state: stopped
Initial processes: 0
Exit state: started
Exit description: Starting stopped PASOE instance myInstance
Exit processes: 14644 11620
Exit status: 0
Exit errors:

proenv>
```

4. Confirm that the HealthScanner is running. Use the following:

   ```
   tcman env
   ```

```
proenv>tcman env
catalina home:     C:\Progress\OpenEdge\servers\pasoe
catalina base:     c:\OpenEdge\WRK\myInstance
catalina tmpdir    c:\OpenEdge\WRK\myInstance\temp
catalina pid:      c:\OpenEdge\WRK\myInstance\logs\catalina-myInstance.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:      myInstance
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: 14644
window title:      PAS-myInstance-41768
security model:    developer
service:           false
servicename:


proenv>
```

5. The `manager health port: 8899` identifies the port for the HealthScanner.

Access HealthScanner data

The following table list ways to request data returned by the HealthScanner probes:

Request	Return value
http://hostname:8899/health	Returns a status code. Typically, `200` when healthy and `500` if health falls below a threshold.
http://hostname:8899	Returns a heat map to represent weight and status of various probes.
http://hostname:8899/health?view=summary	Returns JSON formatted summary of health score.
http://hostname:8899/health?view=details	Returns JSON formatted details of health probes.
instance/bin/oehealth	Uses OEJMX to gather the health values. This could be pushed to a raw data collector for use with administrative scripts.

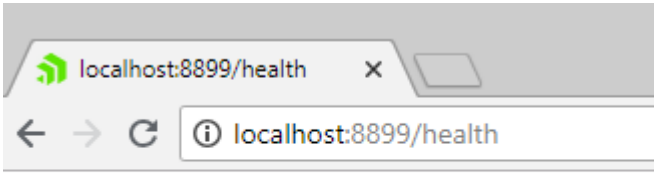
Return the HealthScanner status code

Typically used with the ELB health check, the http://*hostname*:8899/health can be used to return a status code for the server.

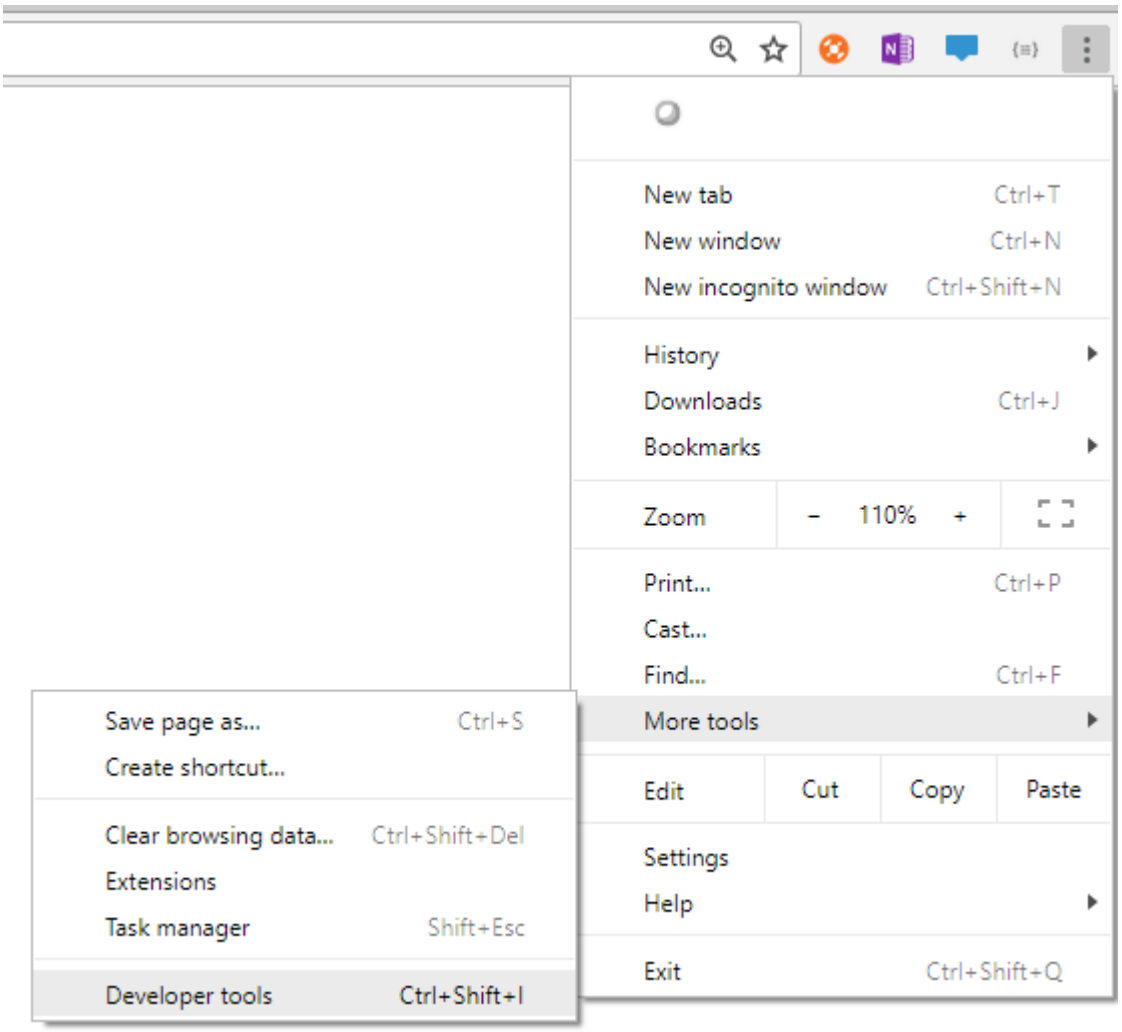
1. Enter the following URL in a browser:

   ```
   http://localhost:8899/health
   ```

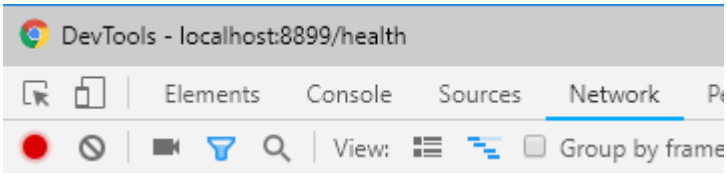
2. A success code of 200 or a blank response is returned depending on the browser shown here:



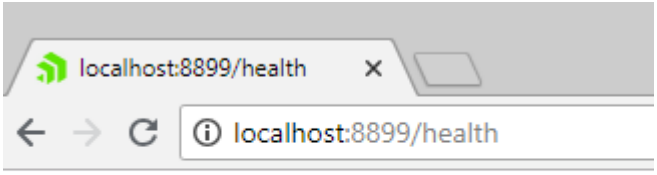
3. To show the status code in Chrome select **…, More Tools, Developer tools** from the toolbar.



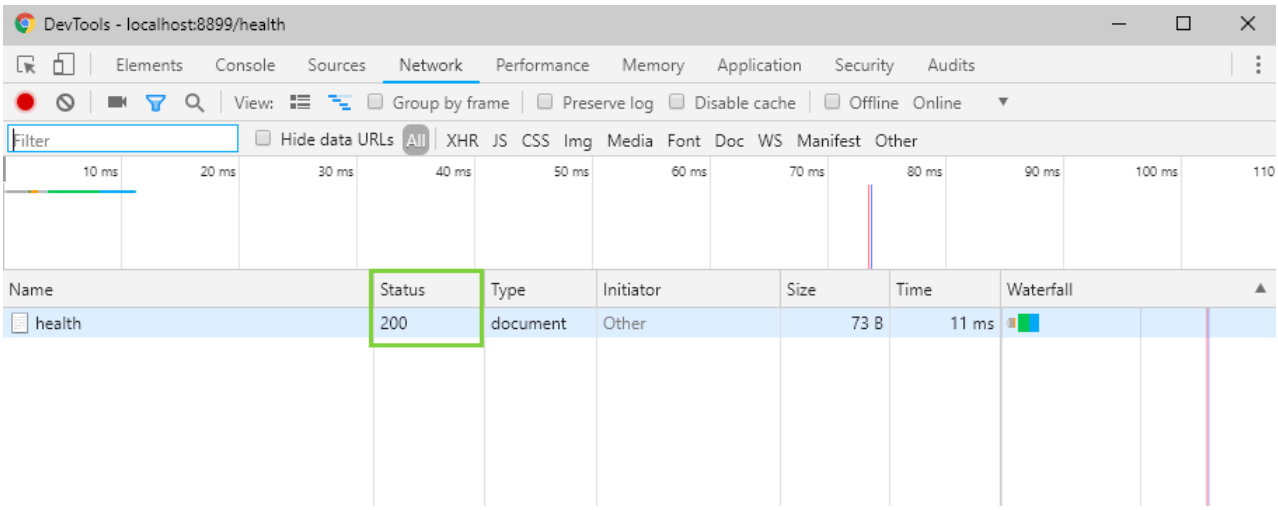
4. Select **Network** from the menu.



5. Refresh the URL.



6. Return to the DevTools browser window, note a **Status** of **200** indicating that the HealthScanner is available.
.

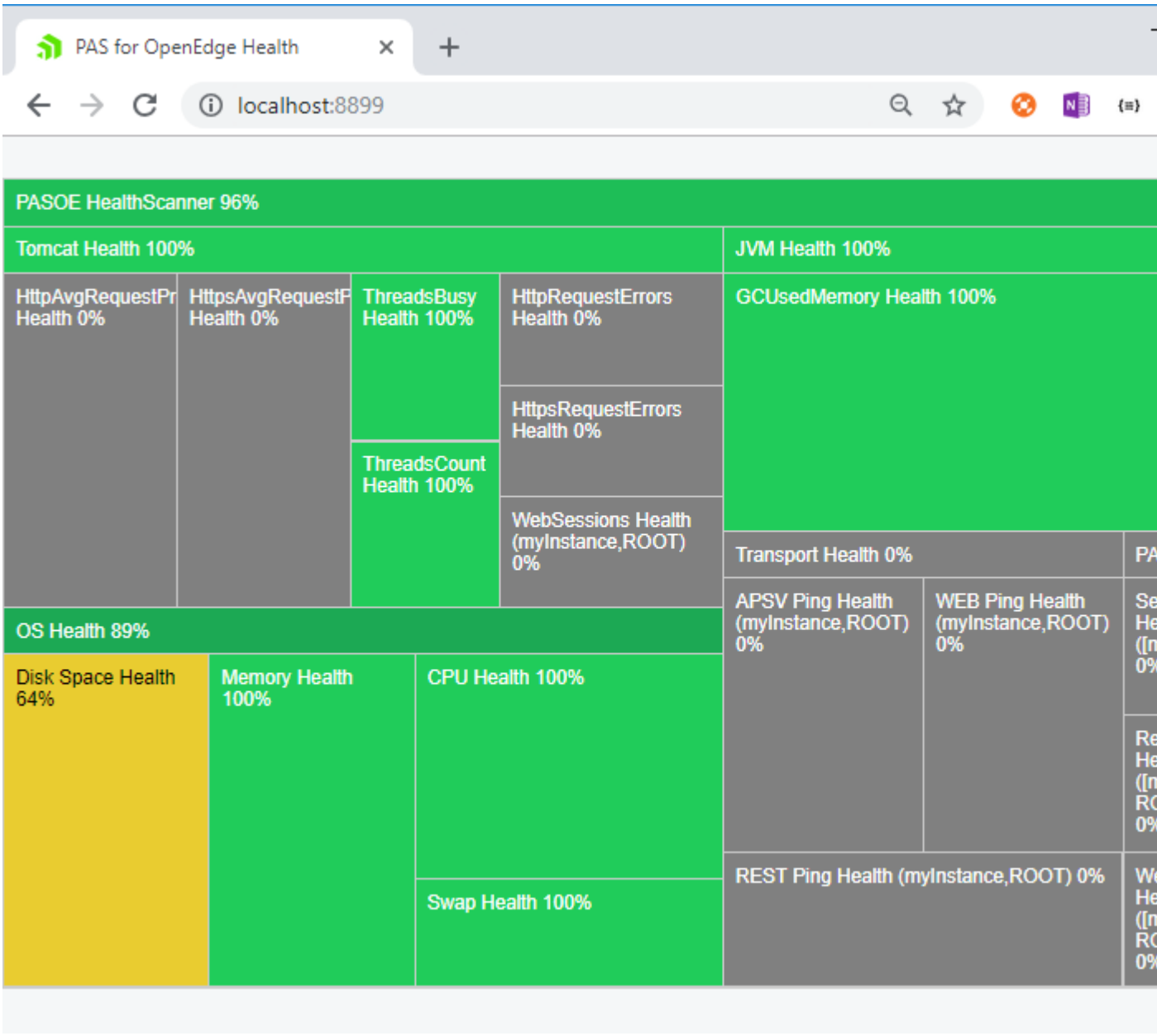


Return a heat map

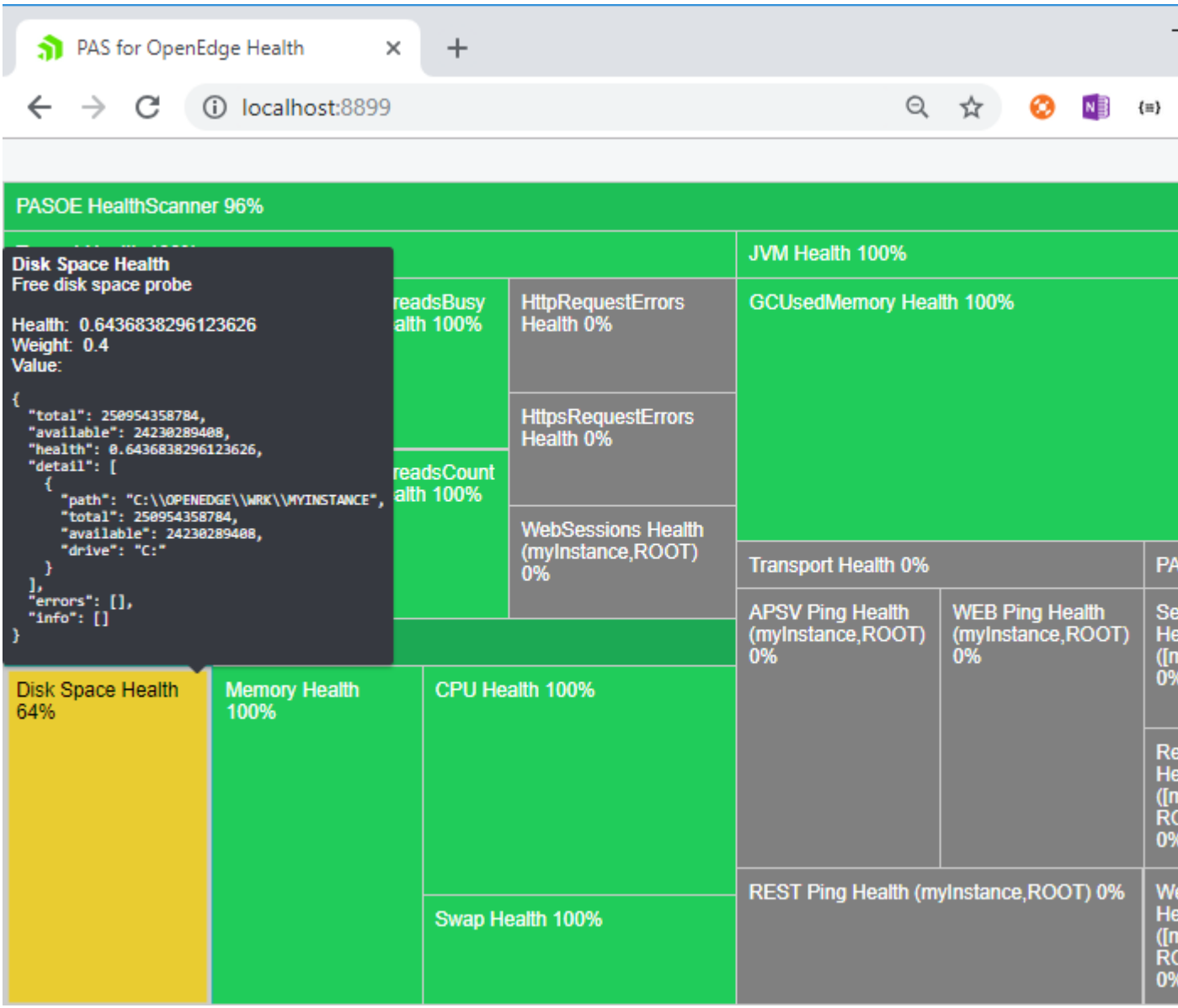
To view the results of all the active probes a graphical representation of the overall health of the server and its underlying components is returned. The display is based on the weighted responses of the probes. The **marginal** and **critical** values are set in the *instance*\conf\health.config file.

Color	Color indicates
Gray	Resource is not currently contributing to the overall health score typically because it has yet to be accessed/activated.
Green	Score is higher than the marginal value.
Yellow	Score is between the marginal and critical value.
Red	Score is below the critical value.

1. In a browser connect to `http://localhost:8899` to poll the load the current heat map. For this request:
   - The overall health is 96%
   - The tomcat health is 100%
   - Green areas are higher than marginal
   - Gray are areas have not been accessed, as requests are made to those areas, the appearance will change.
   - Yellow indicates a warning that the operating system is low on disk space.
   - As the System Administrator you control the thresholds that change the heat map.
   - A critical issue would appear in red, this server has no critical issues.



2. Hover over a particular area. A pop-up displays the detailed JSON data collect from the related probe. The following shows the **OS Health** is at **89%** due to lack of disk space. The **Disk Space Health** is roughly **64%**.



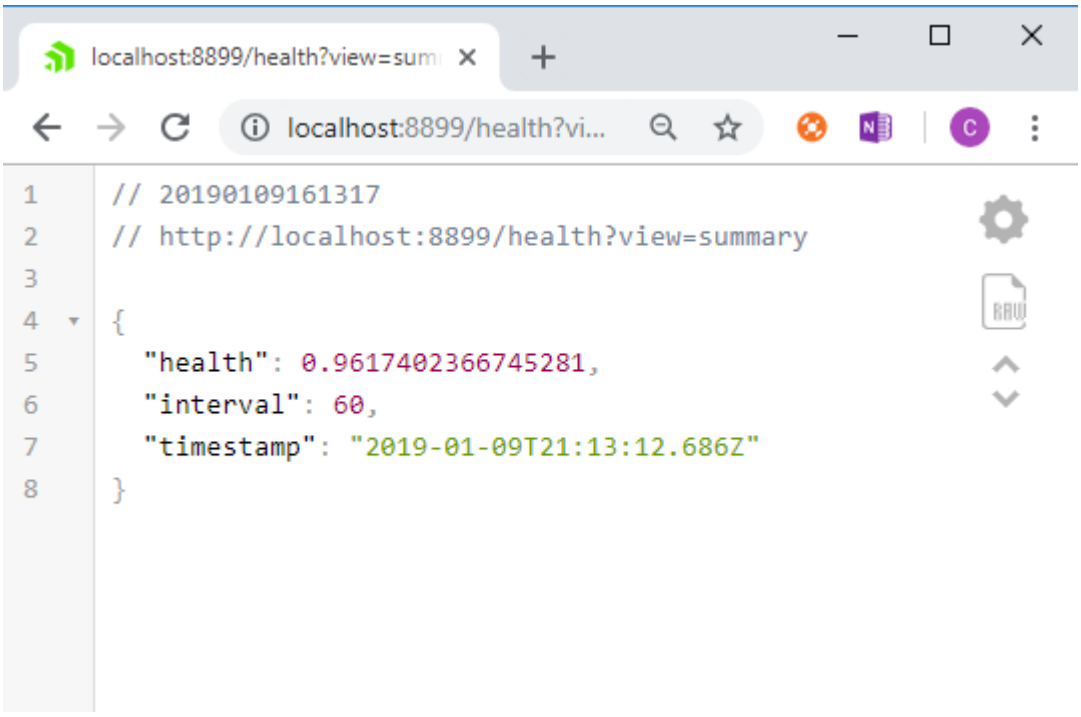
Return a JSON summary data

To return a JSON summary of the overall health score complete the following:

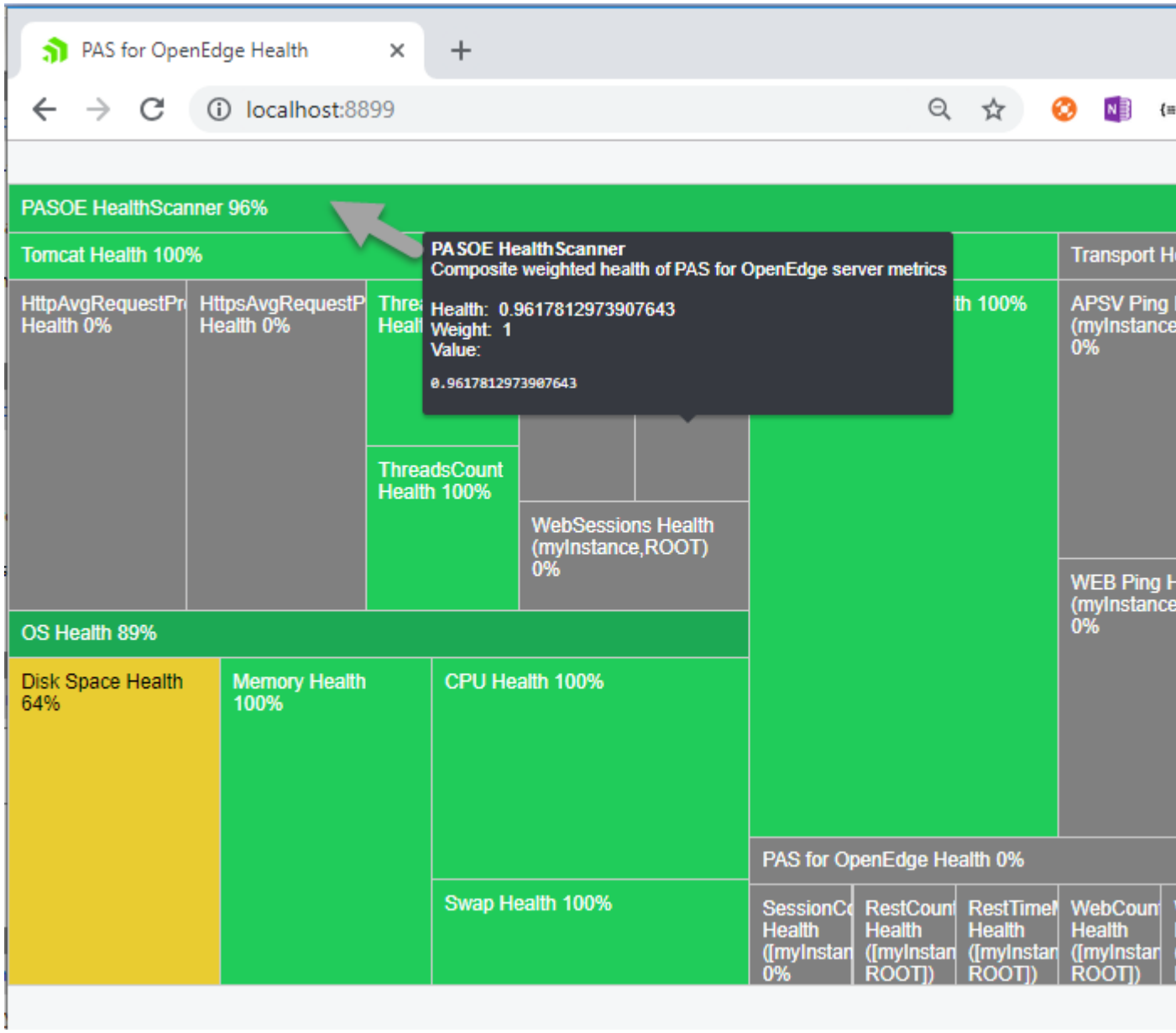
1. Enter the following URL:

   `http://hostname:8899/health?view=summary`

2. Locate the `health` value that represents the overall health of the server. In this example `.95` is the equivalent of **95%**, or good health, based on the probes.

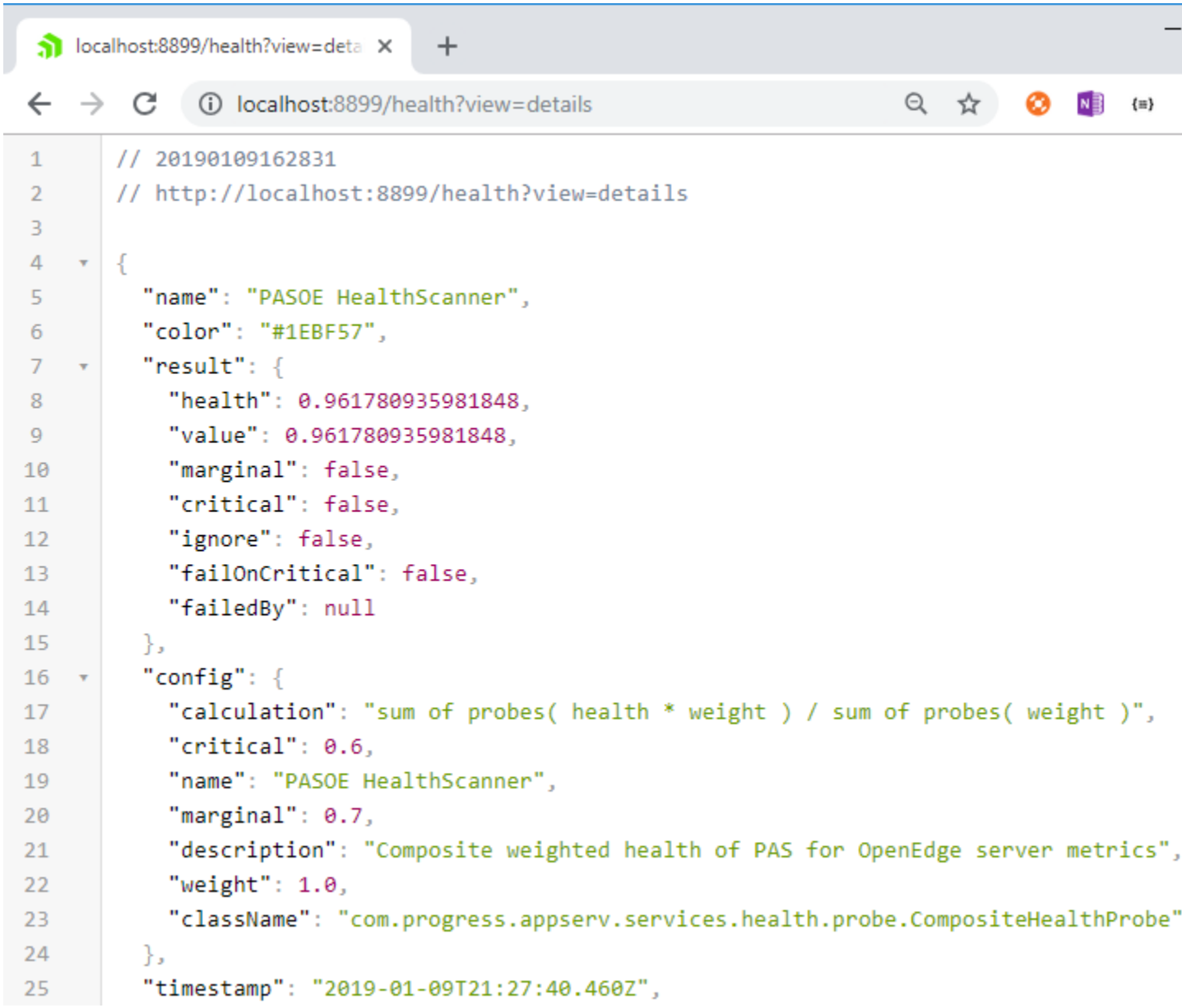

localhost:8899/health?view=sum
localhost:8899/health?vi...
// 20190109161317
// http://localhost:8899/health?view=summary
{
"health": 0.9617402366745281,
"interval": 60,
"timestamp": "2019-01-09T21:13:12.686Z"
}

Note: These values are the same as those displayed in the overall health from the heat map when you hover the mouse over the PASOE HealthScanner.



Return a JSON details data

For a complete JSON representation of all the probes, use http://hostname:8899/health?view=details that returns all the probes including details.



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 JMX (Java Management Extensions) OEJMX connects to an instance using Java objects called Managed Beans or 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.

1. Use a `proenv` command prompt to change to the instance directory. For example:

   ```
   cd instance
   ```

2. Request the the overall health score. Enter:

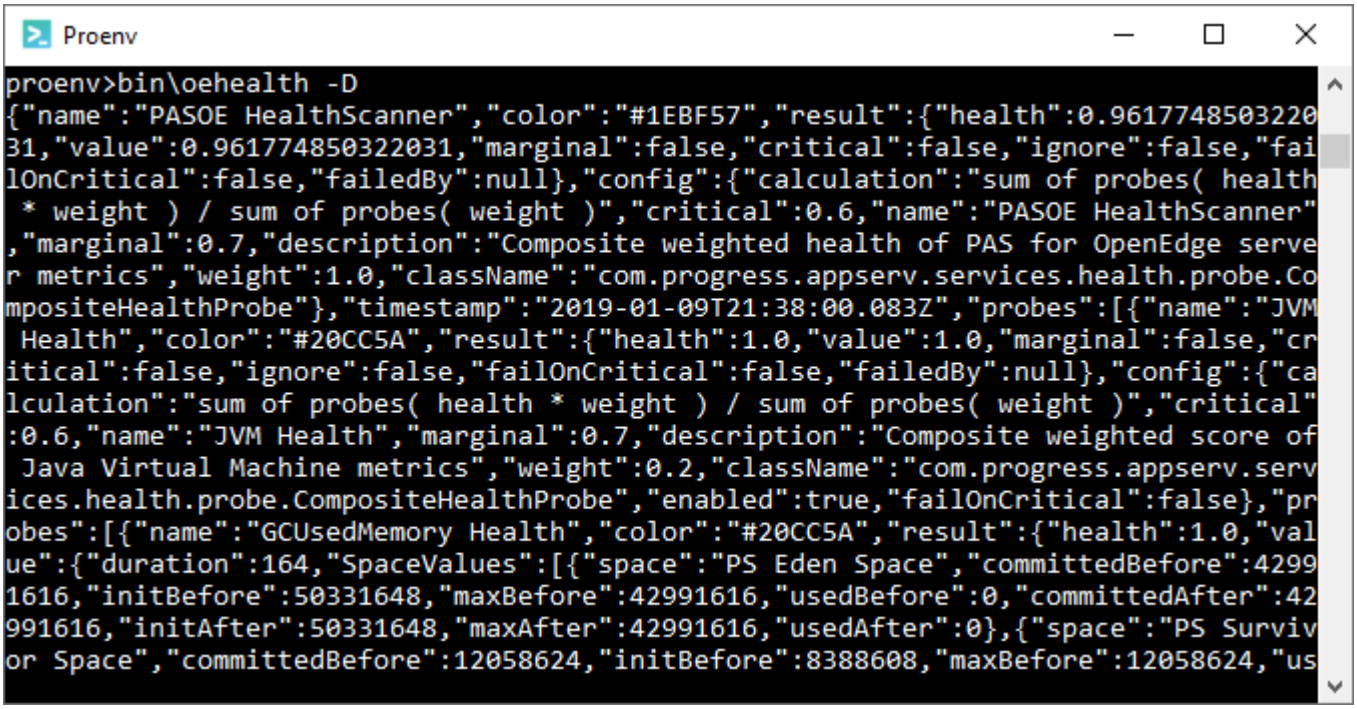
   ```
   bin\oehealth
   ```



Note: This utility can be included in scripts to monitor your server.

3. To display the detailed JSON for the server health. Enter the following:

   ```
   bin\oehealth -D
   ```

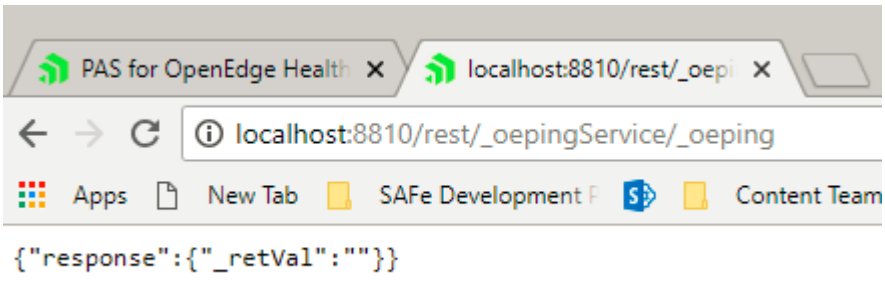


Display results of a healthy REST call

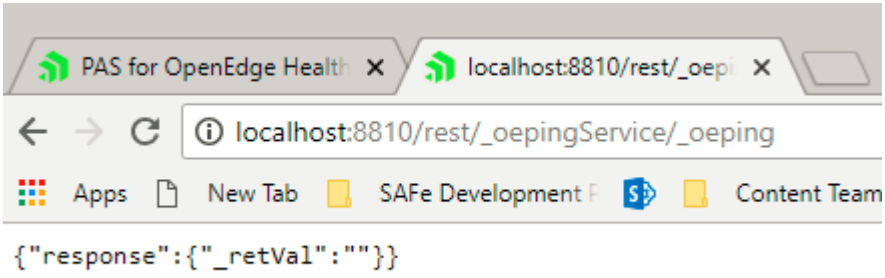
To activate the probes for a REST call, you can use the `_oeping` service shipped with the product. This will turn activate the REST probes, turning those areas from gray to green in the heat map. In a later example, an unhealthy request will be made and different results will be displayed.

1. In a separate tab of the browser make a ping service REST call. Enter the ping request:

   ```
   localhost:8810/rest/_oepingService/_oeping
   ```



2. Confirm the `"_retVal":""` was returned.

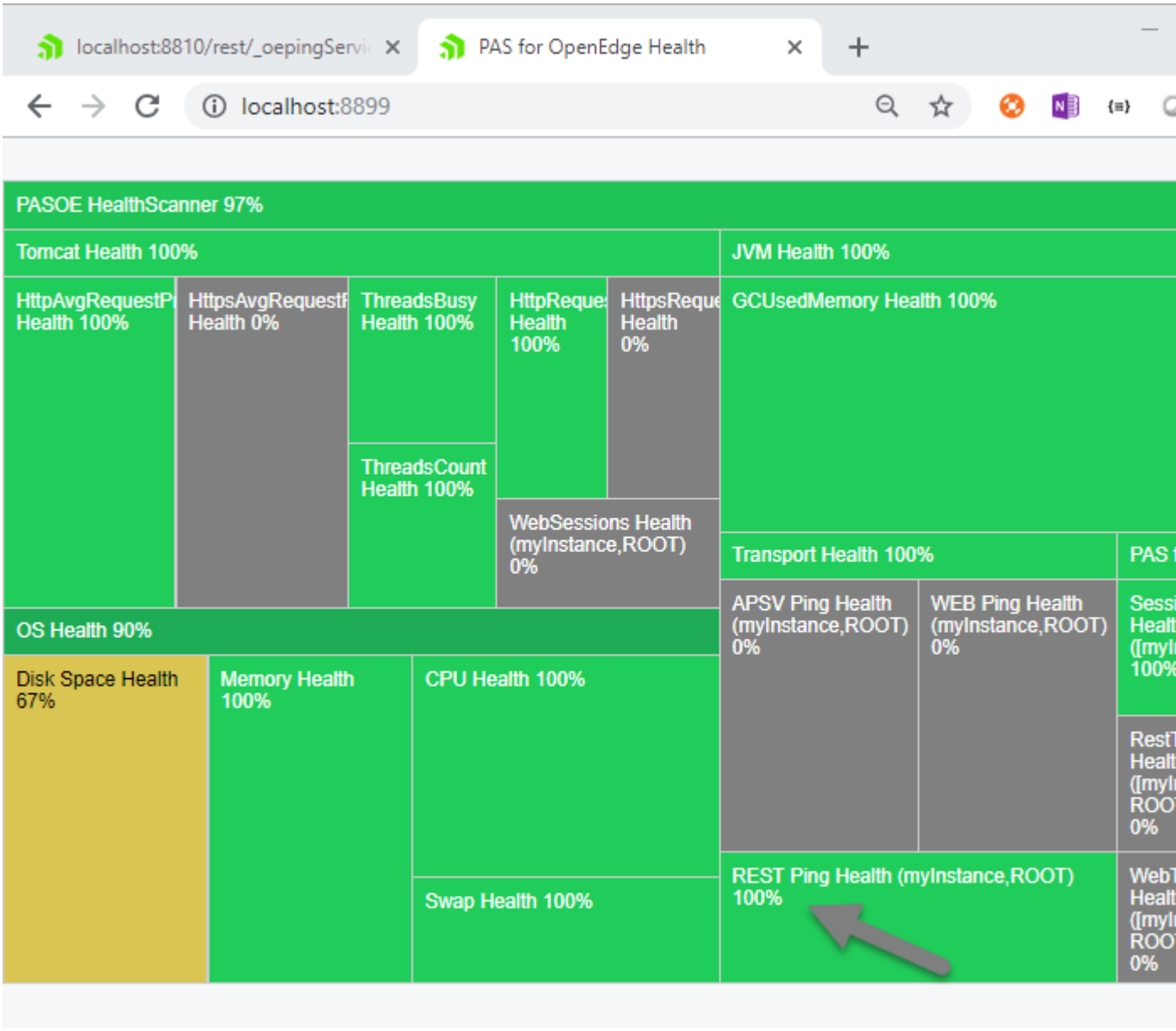


Note:

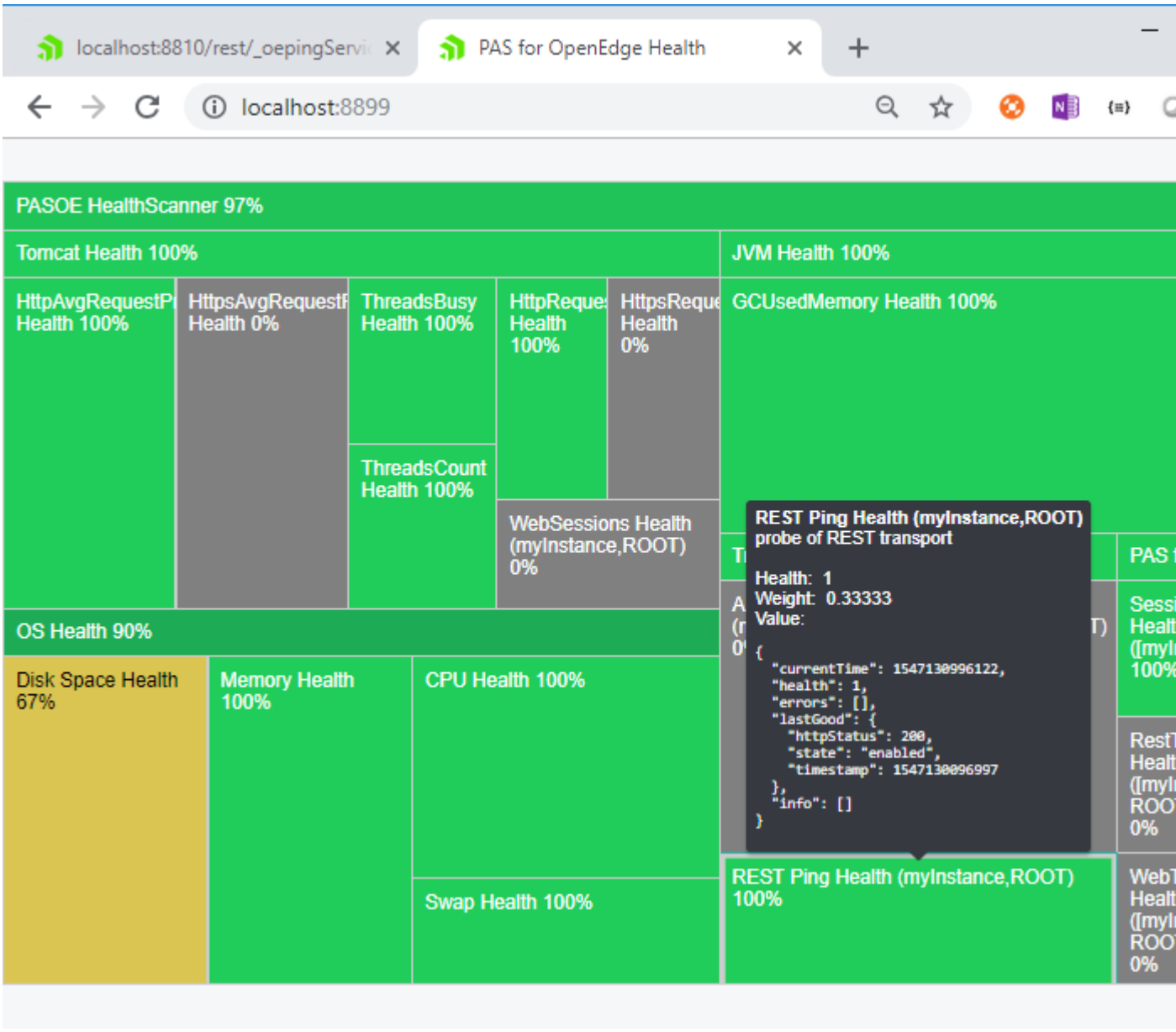
For JSON formatted return values, add the appropriate JSON formatting plugin for your browser.



3. Return to the tab for the HealthScanner and refresh the view to show the activity for the REST service. Notice the changes in the lower right for **REST Ping Health**. The color changed from gray to green and the health is 100%.



4. Hover over the map to display details for that area.



Note: The `health` as `1` or **100%** for the **REST Service Health** and the **Weight** that this has in the overall health, `0.33333`.

Display results of an unhealthy REST call

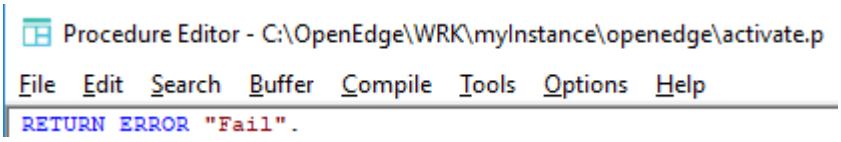
In this section, you introduce an error to see how this affects the health score of the server. Edit the `openedge.properties` file and add an `activate.p` procedure as the **session activate** procedure. This procedure is run when a session is started by REST call. For the purpose of this demonstration, 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`*`\conf` directory to specify a `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 `openedge.properties`.
3. Use the editor to create an `activate.p` procedure that returns an error and string.

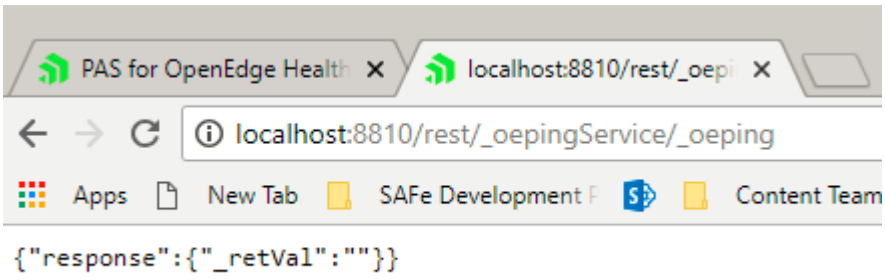
```
RETURN ERROR "Fail".
```



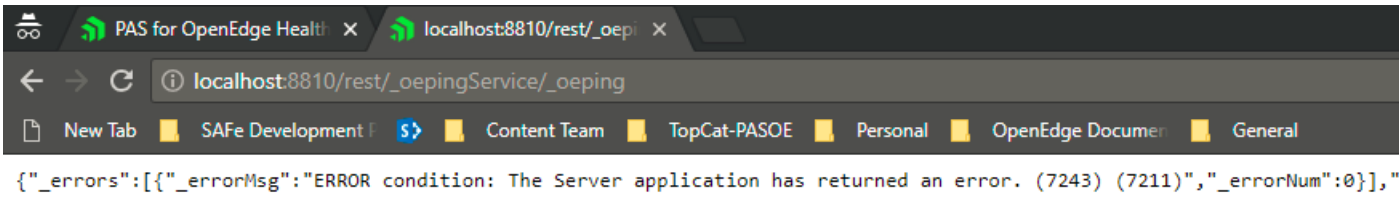
4. Save the file to the *instance*`\openedge\activate.p` which puts the file in the PROPATH for the instance.
5. Restart the server. Enter the following:

```
tcman pasoestart -restart
```

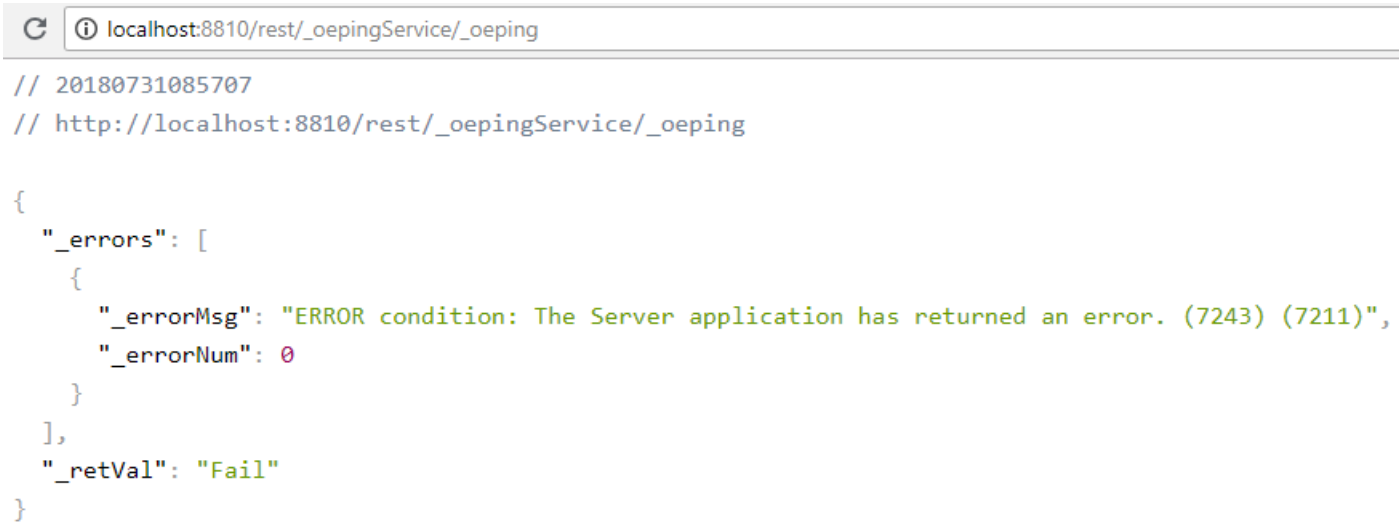
6. Test the REST service again.



7. This time there will be an error introduced by the `activate.p`, the unformatted output is this:



The formatted output is this:



8. Return to the tab for the HealthScanner. Refresh the view to show the **Server Health** is now **marginal**. The **Transports** score has changed to **critical** because the REST Ping returned an **httpStatus** of **500** shown when hovering over **REST Ping Health** on the heatmap.



Note: As a result, the **PASOE HealthScanner** score lowered to **70%**.



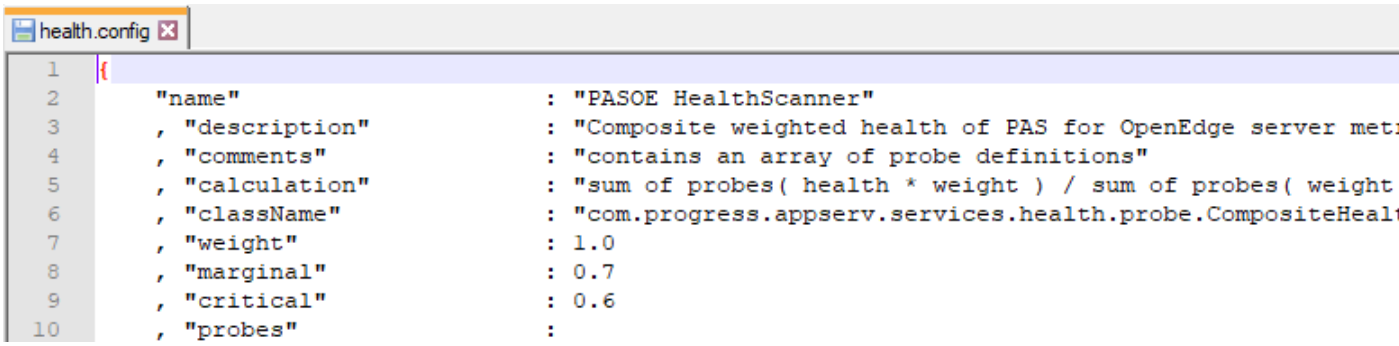
9. Modify the `openedge.properties` file located in the *`instance`*`\conf` directory to reset the value for `sessionActivateProc=` to the original blank value to avoid any future problems.

Note: The HealthScanner acts as the warning light not a diagnostic tool for analyzing the issue. The next steps would be to review the log files to discover the underlying cause of the lowered health score.

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, the marginal and the 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.

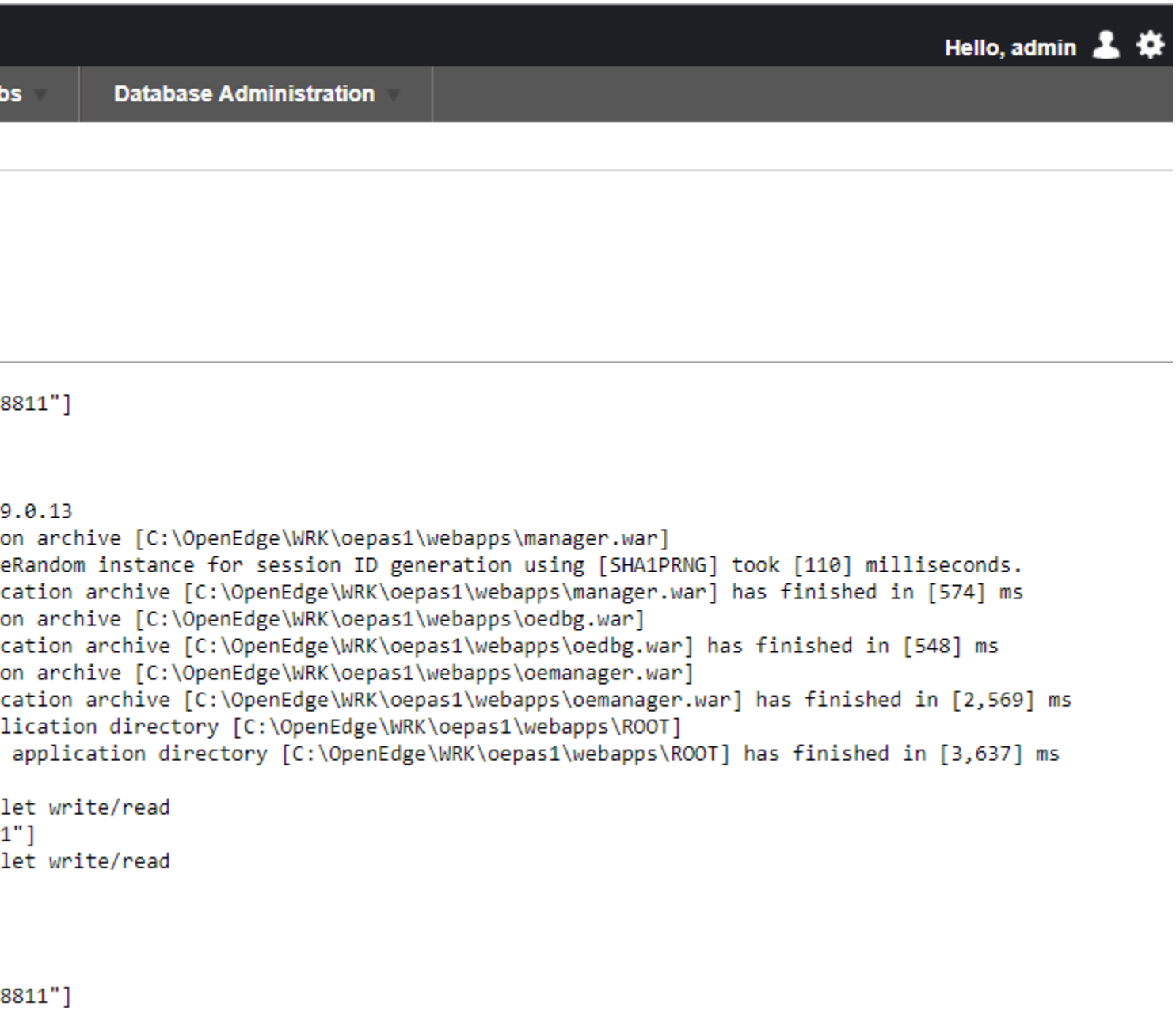
3. Close the file without saving changes.
4. To learn more about the available settings and specific probes, review the *instance-name*\conf\health.config.README file.

```
# -------------------- 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.
```

Use the log file viewer from OpenEdge Management

The log file viewer allows system administrators to examine various Tomcat log files through a graphical HTML interface. You can use the log file viewer from OpenEdge Management.

The following figure shows the Log File Viewer displaying the contents of the `catalina.date.log` in OpenEdge



The figure shows the initialization of the Tomcat server along with OpenEdge web applications.

For more information, see Accessing and reviewing PAS for OpenEdge log file data.

Troubleshoot problems with an instance

When debugging 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

1. 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.
2. Scan the file for the time of the error. Time is based on the system time of the PAS for OpenEdge Tomcat process.
3. If no errors are listed there: go one level higher to the *abl-app-name*.*date*.log.
4. Scan the file for the time of the error.
5. If no errors are listed within these files at the expected time, search higher still to see if an error is recorded in either 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. This is described in Configure logging in PAS for OpenEdge.
7. You can specify additional Log Entry Types in openedge.properties for your agent. For example:
   - 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
8. If memory consumption for the multi-session agent is continually increasing, there may be memory leaks in the application code. See the following articles for more information:
   - How to check for leaked dynamic objects?

- Sample Code for Analyzing PASOE Agent Log for Possible Memory Leaks

Note: If an item shows up as a potential leak, that does not mean that it is actually the problem. These articles simply report on objects that are sticking in memory.