Progress®



Migrate Classic AppServer Applications to PAS for OpenEdge

OpenEdge®

Table of Contents

Copyright....5
Preface....7

Learn About Migrating Classic AppServer Applications to PAS for OpenEdge....9
Differences between classic AppServer, WebSpeed Transaction Server, and PAS for OpenEdge....10

Quick Start: Move Classic AppServer Applications to PAS for OpenEdge....13
Welcome to PAS for OpenEdge....16
Prepare to move from Classic AppServer to PAS for OpenEdge....16
Move classic AppServer code to PAS for OpenEdge....23
Connect clients with new PAS for OpenEdge transports....28
Important details about your new server....32
Next steps with PAS for OpenEdge....34

ABL Application Code Migration....35
ABL application structure....36
Migrate classic AppServer operating modes....38
Migrate classic state-reset operating mode....40
Migrate classic state-aware operating mode....40
Migrate classic stateless operating mode....41
Migrate classic state-free operating mode....41
Differences in event procedures between classic AppServer and PAS for OpenEdge....41
Multi-session agent Startup and Shutdown procedures....41
Session Startup and Shutdown procedures....42
Connect and Disconnect procedures....43
Activate and Deactivate procedures....44
Database connections in PAS for OpenEdge....46
ABL changes in PAS for OpenEdge....47
Environment settings in PAS for OpenEdge....48
File system directories and paths in PAS for OpenEdge....48
Migrate client connections to PAS for OpenEdge....49
Migrate AIA URLs to use the APSV transport....52
Migrate WSA URLs to use the SOAP transport....53
Migrate REST URLs....54
Use load balancing in PAS for OpenEdge....54

Migrate Server Configuration and Management.....57
Manage server instances.....58
Migrate classic AppServer properties.....60
Migrate classic AppServer properties to PAS for OpenEdge using PASPROPCONV.....65
Migrate multiple classic AppServer properties to PAS for OpenEdge using PASPROPCONV...65
Application packaging and installation.....68
Distribution packaging.....68
Incremental SOAP services.....69
Incremental REST services.....69
Basic web application packaging.....69
PAS for OpenEdge instance packaging.....70
Installation process and updates.....70
Performance and resource tuning.....73
Classic AppServer features not applicable to PAS for OpenEdge 73
Development process tooling.....73
Installation packaging and deployment.....74
Site monitoring and administration tools.....74
Security in classic AppServer versus PAS for OpenEdge.....75
Spring Security in PAS for OpenEdge 75

Migrate Classic WebSpeed Applications.....77

Copyright

© 2021 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

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

DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Defrag This, Deliver More Than Expected, DevReach (and design), Icenium, Inspec, Ipswitch, iMacros, Kendo UI, Kinvey, MessageWay, MOVEit, NativeChat, NativeScript, OpenEdge, Powered by Chef, Powered by Progress, Progress, Progress Software Developers Network, SequeLink, Sitefinity (and Design), Sitefinity, Sitefinity (and design), SpeedScript, Stylus Studio, Stylized Design (Arrow/3D Box logo), Styleized Design (C Chef logo), Stylized Design of Samurai, TeamPulse, Telerik, Telerik (and design), Test Studio, WebSpeed, WhatsConfigured, WhatsConnected, WhatsUp, and WS_FTP are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries.

Analytics360, AppServer, BusinessEdge, Chef Automate, Chef Compliance, Chef Desktop, Chef Habitat, Chef WorkStation, Corticon.js, Corticon Rules, Data Access, DataDirect Autonomous REST Connector, DataDirect Spy, DevCraft, Fiddler, Fiddler Everywhere, FiddlerCap, FiddlerCore, FiddlerScript, Hybrid Data Pipeline, iMail, JustAssembly, JustDecompile, JustMock, KendoReact, NativeScript Sidekick, OpenAccess, PASOE, Pro2, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Push Jobs, SafeSpaceVR, Sitefinity Cloud, Sitefinity CMS, Sitefinity Digital Experience Cloud, Sitefinity Feather, Sitefinity Insight, Sitefinity Thunder, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Supermarket, SupportLink, Unite UX, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

July 2021

Product version: Progress OpenEdge 12.4

Preface

Purpose

The purpose of this content is to introduce the following concepts:

- Migrating classic AppServer applications to PAS for OpenEdge
- Quick Start: Move to PAS for OpenEdge tutorial
- ABL application code migration reference
- Migrate server configuration and management reference
- Migrate classic WebSpeed applications

Audience

OpenEdge users looking to migrate from the classic OpenEdge AppServer to the Progress Application Server for OpenEdge (PAS for OpenEdge).

Organization

Learn About Migrating Classic AppServer Applications to PAS for OpenEdge on page 9

Quick Start: Move Classic AppServer Applications to PAS for OpenEdge on page 13

ABL Application Code Migration on page 35

Migrate Server Configuration and Management on page 57

Migrate Classic WebSpeed Applications on page 77

Documentation conventions

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

1

Learn About Migrating Classic AppServer Applications to PAS for OpenEdge

Migrating an application from the Progress OpenEdge AppServer (classic AppServer) or the WebSpeed Transaction Server to Progress Application Server (PAS) for OpenEdge begins with moving your application to a test environment with a few simple code and property changes. The following materials are designed to help you move your application.

- Quick Start: Move Classic AppServer Applications to PAS for OpenEdge on page 13 is a step by step tutorial for moving a classic AppServer application to a PAS for OpenEdge test instance.
- Migrate Classic AppServer Applications to PAS for OpenEdge is a companion video demonstrating how to move a sample ABL application to PAS for OpenEdge using the APSV transport.
- Migrate Classic AppServer REST Services to PAS for OpenEdge is a companion video demonstrating how to move sample classic AppServer REST services to PAS for OpenEdge using the REST transport.

After you move your application to a test system, you are ready to complete the migration by learning more about PAS for OpenEdge. Because every application is unique, there is no universal advice for migrating to a live production environment. The client types and complexity of your existing server configurations will dictate the migration path of your application.

This content begins by highlighting the differences between PAS for OpenEdge and the existing servers, classic AppServer and WebSpeed Transaction Server. The following content is available for a deep-dive into the best practices for migrating classic AppServer and WebSpeed Transaction Server content to PAS for OpenEdge:

- ABL Application Code Migration on page 35
- Migrate Server Configuration and Management on page 57
- Migrate Classic WebSpeed Applications on page 77

Learn more about PAS for OpenEdge

Use the following resources to help you learn more about PAS for OpenEdge.

Read about PAS for OpenEdge:

- Learn about Progress Application Server for OpenEdge, for a detailed introduction
- Manage PAS for OpenEdge, for an introduction to the overall administrative framework including security, tuning, and monitoring options.
- Learn about PAS for OpenEdge and OpenEdge Management, for configuration recommendations including using OpenEdge Management
- Progress Application Server for OpenEdge: Machine Sizing Guide, for details on how to size your PAS for OpenEdge machine for use in production.

Take a course:

- *Progress Application Server for OpenEdge Administration (1000-095)* course available at https://www.progress.com/services/education/openedge/pas-for-openedge-administration

Join the community:

- Ask a question: https://community.progress.com/community_groups/openedge_development
- Download whitepapers on advanced topics and other resources: https://community.progress.com/community_groups/openedge_general/m/documents

For details, see the following topics:

- Differences between classic AppServer, WebSpeed Transaction Server, and PAS for OpenEdge

Differences between classic AppServer, WebSpeed Transaction Server, and PAS for OpenEdge

Differences between PAS for OpenEdge and Progress AppServer

The following list presents the key points that will influence your migration plans.

- **PAS for OpenEdge is a web server platform.** PAS for OpenEdge is a unified web server platform built on Apache Tomcat web server technology and enhanced by OpenEdge development to manage the specific needs of OpenEdge clients. Because the underlying technology is different, there are differences in how you install, configure, connect, and run applications using PAS for OpenEdge
- **Licensing is different.** PAS for OpenEdge is available as two separate product licenses:
  - The *Progress Development Application Server for OpenEdge* is a web server for **developing and testing** OpenEdge applications.
  - The *Progress Production Application Server for OpenEdge* is a secure web server for OpenEdge **production application deployment**.
- **Applications run in an instance.** Unlike the classic AppServer, applications do not run in the PAS for OpenEdge installation directory. Instead, applications run in one or more instances that you create based on the installation template. This makes future upgrades easier because changes made in an instance can be kept when you upgrade to a new release. Additional backup and restore steps are not required for upgrading classic AppServer applications.

- **Special adapters are no longer required.** With PAS for OpenEdge, there are no special adapters (WSA or AIS) to support different OpenEdge client types. Instead, instances support four transport types to connect all existing and future client types.
  - **APSV transport** is for ABL clients, Java clients, or .NET OpenClients.
  - **REST transport** is for existing REST, mobile, and web clients.
  - **SOAP transport** is for SOAP web service clients.
  - **WEB transport** is for WebSpeed and new web service clients.

  When a given client type connects to an instance, the URL must contain the transport required for that type of client. These transports are important when you configure security. You can enable and disable a transport for a given ABL web application to secure specific applications.

  Note: For OpenEdge REST web services implemented on the AppServer, you can migrate the AppServer business application to a PAS for OpenEdge instance so that existing clients of OpenEdge REST and Data Object Services can connect and send requests using the same resource URIs. New REST applications should be implemented with the WEB transport to access more REST methods.

- **Clients connect differently.** Client connections are standardized on HTTP or HTTPS URL connections to a web application (an OpenEdge ABL web application derived from `oeabl.war`). The OpenEdge ABL web application, also known as an ABL web application, then manages the execution of all server ABL code on behalf of all supported OpenEdge client types. Your applications no longer use direct or native mode connections to an instance from an ABL, Java, or .NET Open client as they did with the classic AppServer.
- **Client requests are distributed differently.** Each deployed ABL web application includes a **session manager** that handles all client connections to the web application, and distributes client requests to the server's available multi-session agents, which in turn pass each request to an available ABL session. Each configured **multi-session agent** is a separate operating system process that maintains a pool of ABL sessions that are used to execute all client requests sent to that agent. Depending on the application, each multi-session agent process can run requests in multiple ABL Sessions at one time, allowing a single agent process to handle multiple requests from one or more clients in parallel. Therefore, compared to the classic AppServer, where every ABL Session runs in its own agent (OS process), the multi-session agent process can handle the same client load using fewer OS resources.
- **Two application models replace four operating modes.** Unlike the classic AppServer, which supports four different operating modes, PAS for OpenEdge instances support two operating modes, referred to as **application models**. The supported models are:
  - **Session-free model** which runs almost exactly like the classic AppServer's **state-free** operating mode
  - **Session-managed model** runs similarly to the classic AppServer's **stateless** operating mode.

  Note: You can emulate **state-reset** or **state-aware** operating modes with code changes to the connect and disconnect configuration procedures, referred to as **event procedures** for an instance. For more information, see Migrate classic AppServer operating modes on page 38.

- **Client connections set the application model, not the server.** You do not configure an instance to support a single application model as you do with the classic AppServer. Each client connection identifies the application model that the connected ABL web application uses to service requests from that connected client. This is the same way that a client identifies the session mode when it connects to the classic AppServer. The benefit of this type of connection is that any available session can handle requests from any given client according to the application model that it requires. When you migrate ABL code to an instance from a classic AppServer application, the code should run on that instance with very few (if any) changes, as described in ABL Application Code Migration on page 35. If you try to migrate multiple classic AppServer applications running in different operating modes to a single instance, you must make more

code changes, especially to an instance's event procedures, in order to handle the context management for any changes in the application model and binding state for each client connection.

- **Client requests are processed simultaneously within the same agent OS process.** To execute client requests, each multi-session agent maintains two pools within its single process space: a pool of ABL Session contexts and a pool of ABL Virtual Machines (AVMs, which are threads in the agent process) that execute the ABL code. When a multi-session agent receives a client request, it associates an available ABL Session context with an available AVM, and the AVM executes the ABL code to handle the request in the associated session context. When the AVM finishes executing the client request, both the AVM and its associated ABL Session context are returned to the agent's respective pools. The newly available ABL Session retains all context (global data) established for the most recent client request unless it is explicitly cleared by the application during the handling of the request.
- **Client connections are unbound by default.** Client connections are unbound to a session connection to allow efficient reuse of server resources. If your application requires a client connection to be bound to a session connection, then see "Manage context for bound and unbound session-managed connections" in *Application Development with PAS for OpenEdge*.
- **Load balancing is no longer handled by the NameServer.** There is no NameServer or Unified Broker architecture in PAS for OpenEdge because each instance is a web server. However, you can achieve fault tolerance and load balancing across multiple instances that support a single session-free business application by registering the instances with an industry-standard HTTP load balancer. Your clients connect to the load balancer using a URL that identifies the same ABL web application and the appropriate connection transport supported by all the registered instances. Depending on the load balancing service, it can then delegate client requests to the available instances using different load balancing policies.

Differences between PAS for OpenEdge and the WebSpeed Transaction Server

PAS for OpenEdge provides a special compatibility web handler that has the following basic advantages over the WebSpeed Transaction Server:

- WebSpeed applications on a PAS for OpenEdge instance have a more integrated architecture as compared with the WebSpeed Transaction Server because both the web server and the WebSpeed Transaction Server are combined in a single instance.
- WebSpeed on an instance is more efficient than WebSpeed on the Transaction Server regarding the management and the availability of the agents that handle client requests.
- WebSpeed Transaction Server supports only the GET and POST HTTP verbs. WebSpeed on an instance supports all standard HTTP verbs.
- WebSpeed on an instance supports event procedures that were not supported on the WebSpeed Transaction Server.
- PAS for OpenEdge includes support for multiple servers in a single instance. You do not need to configure and run a separate web server, WebSpeed Transaction Server, and classic AppServer.
- An instance shares a single security context among the WEB transport that supports WebSpeed and all the other transports (REST, SOAP, and APSV).
- PAS for OpenEdge provides the option to create custom web handlers to address application-specific needs.

A detailed introduction to PAS for OpenEdge can be found in Learn about Progress Application Server for OpenEdge*Learn about Progress Application Server for OpenEdge*.

2

Quick Start: Move Classic AppServer Applications to PAS for OpenEdge

System administrators can move existing OpenEdge Application Server (classic AppSever) applications to PAS for OpenEdge production and staging environments using the tools provided by OpenEdge and the guidance from this tutorial and the PAS for OpenEdge documentation.

The goal of this Quick Start tutorial is to enable you to run a classic AppServer application on a PAS for OpenEdge instance and update an ABL client connection to use the new APSV transport to access that application. You can use your existing classic AppServer code or the sample code provided.

Sample starter code can be found in the Progress OpenEdge General community: https://community.progress.com/community_groups/openedge_general/m/documents/3775

This Quick Start tutorial does not cover application modernization, performance tuning, application monitoring, or securing your instance for full production. Additionally, this Quick Start tutorial does not highlight all the latest features available in PAS for OpenEdge. Links to this additional information can be found in the Next steps with PAS for OpenEdge on page 34 topic.

Prerequisites

To complete the steps in this tutorial, you should have OpenEdge 12 or later installed on your Linux or Windows 64 machine. You can run the migration using OpenEdge 11.7.3 or later, but you will not be able to use the latest PAS for OpenEdge features.

You must also have the `ubroker.properties` file and run-time code for your classic AppServer application and access to the OpenEdge database. You should also have Progress Developer Studio or Progress Procedure Editor to run the client-side code, but you can also run the code in the Proenv command-line utility.

To complete this Quick Start tutorial, you should have:

- System permissions to install and run scripts
- Access to existing classic AppServer application files
- Classic AppServer configuration knowledge
- A PAS for OpenEdge Development or Production license
- Basic ABL development skills

Files and file structure for the tutorial

You can complete the steps to this Quick Start tutorial using your own classic AppServer application files if you have full access to migrate them to a new server, or you can download the `classicAppServerCode.zip` file from the Progress OpenEdge General Community and use this sample application to test the procedure.

The `classicAppServerCode.zip` contains the following files, to which your classic AppServer application may have analogues:

- `ubroker.properties`—Properties file for the classic AppServer that is located in the *`OpenEdge-Installation-path`*`/bin` directory.

  This example `ubroker.properties` file contains properties for a classic AppServer application called `app_customername_prod`, which will be migrated to PAS for OpenEdge.

- `ServerGetCustNameSample.p`—Sample ABL server code that returns a customer name from the Sports2020 database to a client.
- `ClassicClient.p`—Client-side ABL code that accesses the server-side ABL code using the classic AppServer connection string.
- `PASforOpenEdgeClient.p`—Client-side ABL code that accesses the server-side ABL code using the PAS for OpenEdge connection string.
- `databases.pf`—Parameters file that accesses the Sports2020 database at instance startup. The database can also be accessed directly using a database connection string, but it is a best practice to use a `.pf` file that can remain in a static location even if a database changes.
- `classicREST/classicRESTService.paar`—Sample classic AppServer REST application service definition file. To be used in demonstrating how to migrate classic REST applications to PAS for OpenEdge.
- `classicREST/getCustomer.p` and `classicREST/getAllCustomers.p`—Server-side ABL code that supports the sample `classicRESTService.paar` service definition file.
- `events/session_startup.p` and `events/session_shutdown.p`—Sample startup and shutdown event procedures that write messages to the PAS for OpenEdge application agent log file.

In this tutorial, you will copy a database from the OpenEdge installation directory ($DLC) to a working directory ($WRKDIR). You will also create a PAS for OpenEdge instance in the working directory.

If you are running the sample files, the working directory will be populated with the following file structure as you proceed through the tutorial:

```
WRK

    classicAppServerCode
        ClassicClient.p
        databases.pf
        PASforOpenEdgeClient.p
        ServerGetCustNameSample.p
        ubroker.properties

        classicREST
            classicRESTService.paar
            getAllCustomers.p
            getCustomer.p

        events
            session_shutdown.p
            session_startup.p

    database
        sports2020.b1
        sports2020.d1
        sports2020.db
        sports2020.lg
        sports2020.lic
        sports2020.lk
        sports2020.st
        sports2020_10.d1
        sports2020_20.d1
        sports2020_30.d1
        sports2020_40.d1
        sports2020_50.d1
        sports2020_60.d1
        sports2020_70.d1
        sports2020_80.d1

    myProdInstance
        ablapps
        bin
        common
        conf
        hcapps
        logs
        openedge
        temp
        webapps
        work
```

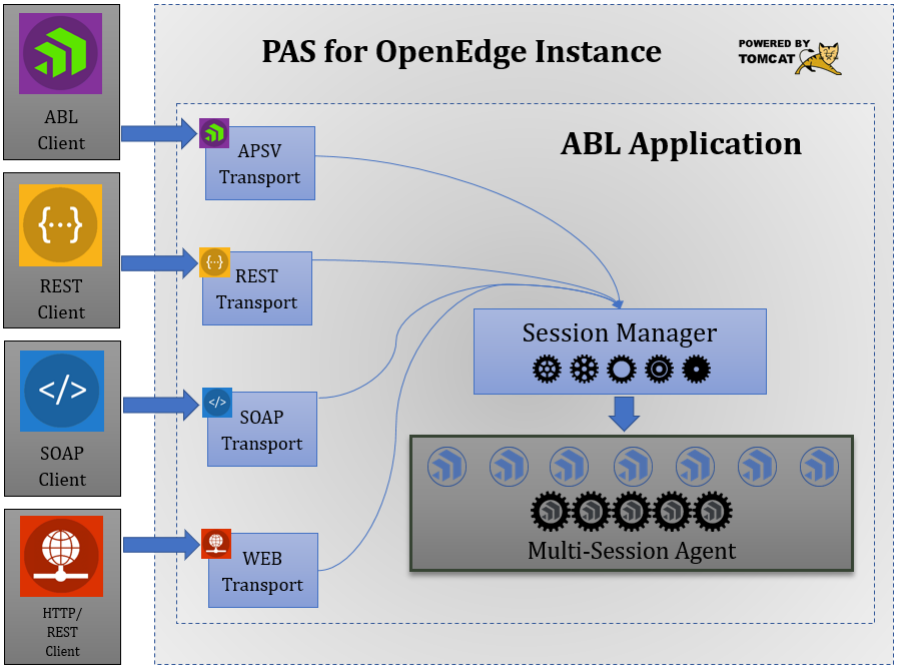
For details, see the following topics:

- Welcome to PAS for OpenEdge
- Prepare to move from Classic AppServer to PAS for OpenEdge
- Move classic AppServer code to PAS for OpenEdge
- Connect clients with new PAS for OpenEdge transports
- Important details about your new server
- Next steps with PAS for OpenEdge

Welcome to PAS for OpenEdge

As a system administrator of a classic AppServer application, it's time for you to move those applications to a production staging instance of the PAS for OpenEdge for testing. This Quick Start tutorial gets your applications running and connects your clients to the new server. After completing the steps in this tutorial, review the Next steps with PAS for OpenEdge on page 34 topic, where you will find additional links for modernizing, securing, and tuning your new server for production.

PAS for OpenEdge is a **unified web server** that does not require special adapters. All client types send requests to your ABL server code using **one of four transports**:



The **Session Manager** distributes those client requests to an underlying **Multi-Session Agent** which ensures efficient use of your server resources. Built on Apache Tomcat web server technology, and enhanced by OpenEdge development to manage the specific needs of your OpenEdge clients, you get the best of both worlds. Progress development applies Apache Tomcat updates that ensure compliance with industry standards, like **Spring security**, while supporting all client types in a single server with no additional products, using easy to configure properties files.

A PAS for OpenEdge instance is created with a predeployed ROOT ABL web application (located in the *instance-name*`/webapps` directory) and an ABL application with the same name as the instance in the *instance-name*`/ablapps` directory. In this tutorial, you will deploy code into the ROOT ABL web application.

Prepare to move from Classic AppServer to PAS for OpenEdge

Complete the following steps to setup the server and adjust its properties to support your application code:

1. Install OpenEdge products.
2. Create a PAS for OpenEdge instance.
3. Generate an `oemerge` file using the paspropconv utility.
4. Move the `setenv` script to the *`instance-name`*`/bin` directory.
5. Review and update the `oemerge` file:
   a. Update PROPATH entries in the `oemerge` file (where required).
   b. Update database connections in the `oemerge` file (where required).
6. Merge the changes using the oeprop utility.

1. Install OpenEdge products

OpenEdge offers **development** and **production** licenses for PAS for OpenEdge. For staging a production configuration, install the **production** license which implements stronger security and limits external management of your server instances.

If you are less familiar with the OpenEdge installation process, see Install OpenEdge.

For more information about the differences between development and production instances of PAS for OpenEdge, see Development versus production instances.

2. Create a PAS for OpenEdge instance

After installing OpenEdge, use `pasman create` to create an instance for testing. This instance acts as the **staging** area for deploying your existing application to a production instance before you move to a live environment. While Windows commands are generally shown here, UNIX has equivalent commands that produce similar results. Notes are included where helpful.

1. Open the Proenv application using the Windows start menu, or by locating it at `$DLC/bin/proenv.sh`. Proenv sets the environment variables and paths for running OpenEdge commands.

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

2. Change to the OpenEdge working directory (`$WRKDIR` or `%WRKDIR%`), which is where your new instance is created and related files and logs are stored. In this example, the directory is called `WRK`:

```
proenv>cd %WRKDIR%
proenv>cd
C:\OpenEdge\WRK
```

3. Run the `pasman create` command to create a named instance called **myProdInstance**:

```
pasman.{sh|bat} create -v -p 8817 -P 8818 -m myAdmin:myPwd -Z prod myProdInstance
```

   In this example, the following options are specified:

   - **`-v`**—Specifies verbose console output.
   - **`-p`**—TCP port that listens for HTTP messages. This example uses 8817.
   - **`-P`**—TCP port that listens for HTTPS messages. This example uses 8818.

- **`-m uid:pwd`**—Username and password combination that replaces the default `tomcat:tomcat` used by Apache Tomcat. The example uses `myAdmin:myPwd` for demonstration purposes only.
- **`-Z`**—Identifies security model of the instance. The example uses `prod` to lock the server with production server security settings. This option disables development accounts and transports by default.
- **`instance_pathname`**—Specifies the absolute path to the directory where instance is created and the instance name. The example uses `myProdInstance`. If no path is specified, the instance is created in the current directory. In this example, the instance is created in the current and recommended directory, `WRK`, because no path is specified.

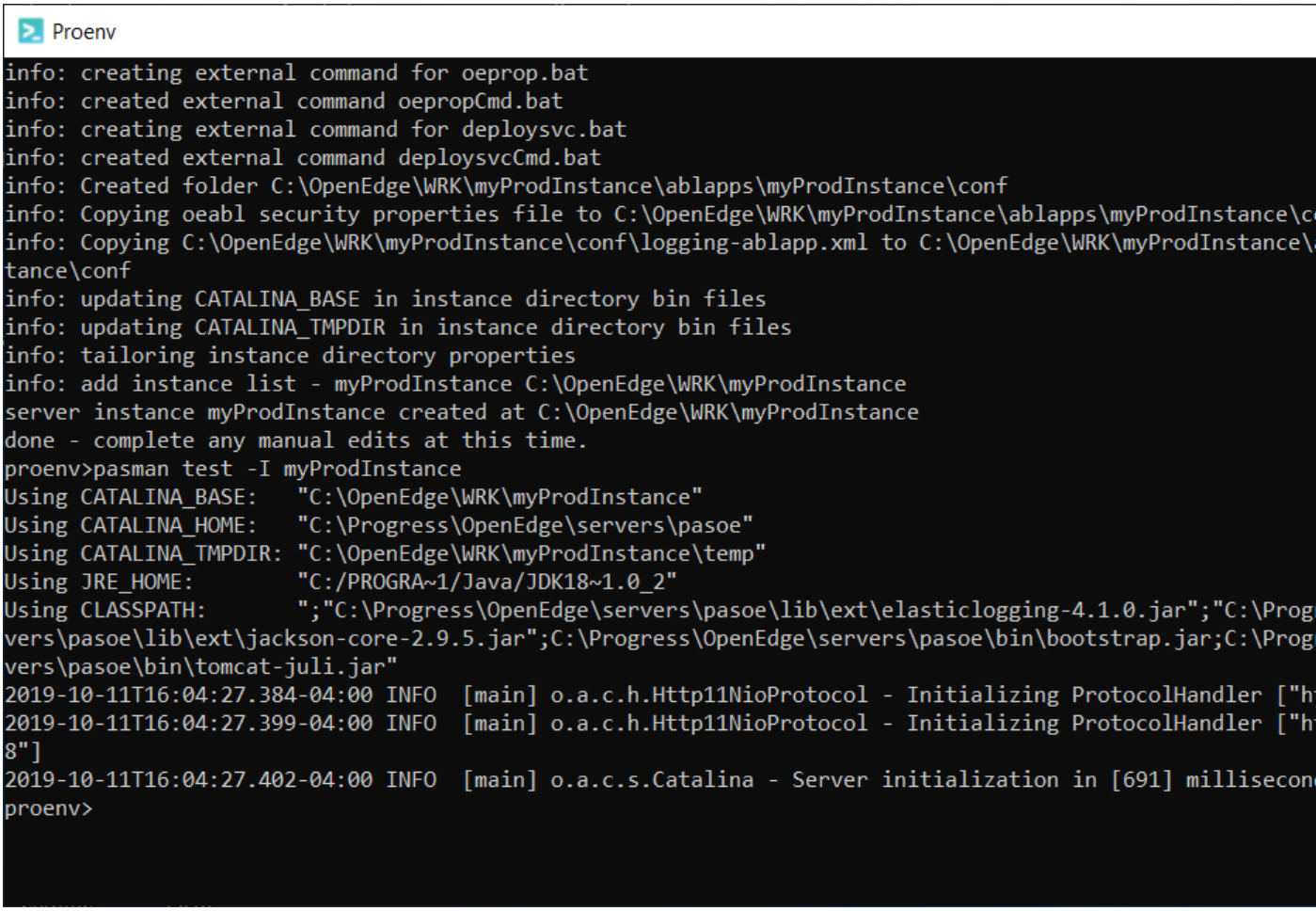
Note: The `pasman` command manages and configures PAS instances using a variety of **actions**. The `create` action creates a new instance from a template. It takes several mandatory and optional **options**. For more information about PASMAN options, type `pasman help create` in the Proenv window, or view Use PASMAN to create an instance.

---

4. Run `pasman test` to confirm that the **ProtocolHandlers** initialize on the port you provided.

```
pasman.{sh|bat} test -I myProdInstance
```

The `pasman test` command displays the PAS for OpenEdge environment variables and confirms that the server can initialize.



3. Create an oemerge file using the paspropconv utility

Your existing `ubroker.properties` must be converted to the new `openedge.properties` format used by PAS for OpenEdge. To complete this step, you can use the `ubroker.properties` file that supports your classic AppServer application, or the `ubroker.properties` file included with the exercise files which supports an example classic AppServer application.

OpenEdge provides a migration tool called paspropconv in the `$DLC/bin` directory of the OpenEdge installation. The paspropconv utility creates a temporary file of changes called *`ubrokername`*`.oemerge` that must be reviewed before merging with the `openedge.properties` file of the new instance.

1. Ensure that you have access to a `ubroker.properties` file for your classic AppServer application. For the purposes of this tutorial, move the `ubroker.properties` file and the contents of the `classicAppServerCode` directory to the working directory using Proenv. For example:

```
move C:\Downloads\classicAppServerCode %WRKDIR%
```

2. Change to the *`instance-name`*`/conf` directory to run the conversion tool in a location where the configuration files are stored. For example:

```
cd %WRKDIR%\myProdInstance\conf
```

3. Run the paspropconv utility which takes an existing `ubroker.properties` file as input and creates a merge file with converted properties and recommendations for your new server. This tool is written in Perl and uses double hyphens (--) to run properly. For this example, the `classicAppServerCode` directory contains existing files from a classic AppServer application:

```
paspropconv
 --ubrokerPropsFile C:\OpenEdge\WRK\classicAppServerCode\ubroker.properties
--ubrokerName UBroker.AS.app_customername_prod
--pasoeAppName myProdInstance
```

   Where:

`--ubrokerPropsFile`	Path to the existing classic AppServer `ubroker.properties` file.
`--ubrokerName`	Fully qualified name of the broker whose properties you are converting. In this example, `UBroker.AS.app_customername_prod`.
`--pasoeAppName`	Name of the new instance. In this example, `myProdInstance`.

   The paspropconv utility generates the following files to the current directory (*`instance-name`*`/conf`):

   - `myProdInstance.app_customername_prod.oemerge`—The merge file for PAS for OpenEdge.
   - `app_customername_prod_setenv.bat`—The environment file for Windows.
   - `app_customername_prod_setenv.sh`—The environment file for UNIX.
   - `paspropconv.log`—The log file for the property conversion.

4. Move the setenv script to the *instance-name*/bin directory

The paspropconv utility generates four files. Move the setenv environment variable file to the /bin directory of your PAS for OpenEdge instance.

1. From the directory where you ran the paspropconv utility (for example, *instance-name*/conf), move the environment variable file for your operating system into the *instance-name*/bin directory. For example:

```
(Windows OS)
>move app_customername_prod_setenv.bat ../bin

(UNIX OS)
>mv app_customername_prod_setenv.sh ../bin
```

5. Review and update the oemerge file

Review the *instance-name*.*uBrokerName*.oemerge file, which is named after the uBrokerName you specified for the conversion. Read each section in the file carefully to understand the changes that will be made to your openedge.properties file.

Note: Every server configuration is different. Read each section in the oemerge file, and adjust your environment accordingly.

The uncommented properties at the bottom of the oemerge file are the properties that will be merged into the openedge.properties file for your new instance. Review and update the uncommented properties based on your server configuration. Pay particular attention to the PROPATH entries and the database connections.

5a. Update PROPATH entries in the oemerge file (where required)

Confirm or update the PROPATH entries to point to any additional or shared code locations. Some subdirectories in your PROPATH may point to your old server. Update these entries to point to the locations on your new server, or ensure you have access to the old file locations and they are specified on your PROPATH. In this sample, you will update the PROPATH to point to the location of event procedures on the PAS for OpenEdge instance.

1. The sample myProdInstance.app_customername_prod.oemerge file indicates that the following startup procedures are required:

```
    sessionShutdownProc=session_shutdown.p
    sessionStartupProc=session_startup.p
```

2. If your classic AppServer configuration used a directory called events to store the startup and shutdown procedures, then copy that directory to your new server. For example:

```
(Windows OS)
proenv>Xcopy /I %WRKDIR%\classicAppServerCode\events
%WRKDIR%\myProdInstance\openedge\events

(UNIX OS)
```

```
proenv>cp -avr $WRKDIR/classicAppServerCode/events
$WRKDIR/myProdInstance/openedge/events
```

3. Reference the location of your new `events` directory in the PROPATH section of your `oemerge` file, or ensure your new server still has access to the old directory location. For example, append the following entry to your PROPATH in the `[AppServer.Agent.app_name]` portion of your oemerge file:

```
[AppServer.Agent.myProdInstance]
    PROPATH=${CATALINA_BASE}/openedge,${CATALINA_BASE}/openedge/events, [...]
```

Note: The `events` directory is one entry in a list of many in the PROPATH. Keep the rest of the entries the same, or change them if you have additional locations for your code.

Note: PAS for OpenEdge does not support accessing the Windows registry. Use environment or Java system variables.

5b. Update database connections in the oemerge file (where required)

You can reference your existing classic AppServer application database by updating your `.pf` file that connects to the database, or by updating the database connection string directly in your `oemerge` file under the `agentStartupParam` property. In this example, you will copy the default OpenEdge database to the working directory and connect to that database using a `.pf` file.

1. If your existing classic AppServer application uses parameter files (`.pf`) to connect to a database or initialize other values, then confirm that those files are available to the new server, or change them to point to the correct file location in the `agentStartupParam` property in `[AppServer.SessMgr.app_name]`. For example:

```
[AppServer.SessMgr.myProdInstance]
    agentLogEntryTypes=ASPlumbing,DB.Connects
    agentStartupParam=-T "C:\OpenEdge\WRK\myProdInstance/temp" -pf
C:\OpenEdge\WRK\classicAppServerCode\databases.pf
```

Note:

The contents of the `.pf` file should specify the database connection string. For example:

```
-db C:\OpenEdge\WRK\database\sports2020.db -H localhost
```

To configure a new database connection using the sample database provided with the OpenEdge installation, follow these steps:

1. In your work directory, create a directory called `database` using Proenv. For example:

```
cd %WRKDIR%
mkdir database
```

2. Run the `procopy` command to copy the sports2020 database from the `$DLC` directory to the `$WRKDIR`:

```
procopy %DLC%\sports2020.db %WRKDIR%\database\sports2020.db
```

Note: Sports2020 is the sample OpenEdge database shipped with OpenEdge 12. If you are running OpenEdge 11.7, use `sports2000` as the database source.

3. (Optional) Edit the `oemerge` file to use the Sports2020 database on agent startup:

```
[AppServer.SessMgr.myProdInstance]
    agentStartupParam=-T "${catalina.base}/temp" -db
C:\OpenEdge\WRK\database\sports2020
```

Note: In the previous example, the `databases.pf` file points to the same location as the direct database connection string in the `agentStartupParam` property. You can keep the `agentStartupParam` assigned to the `databases.pf` file if it contains the correct path to your database.

6. Merge the changes using the oeprop utility

After generating the `oemerge` file using the paspropconv utility and tailoring the file to suit your environment, you can merge the properties with the `openedge.properties` file for your PAS for OpenEdge instance using the oeprop utility.

1. Go to the `/bin` directory of your PAS for OpenEdge instance. For example:

```
cd %WRKDIR%\myProdInstance\bin
```

2. To apply the changes from your `oemerge` file, use the oeprop utility to merge the converted properties into the `openedge.properties` file for your new server. For example:

```
oeprop.{sh|bat} -f ..\conf\myProdInstance.app_customername_prod.oemerge
```

Note: For more information about the oeprop utility, type `oeprop -help` in the Proenv window, or see OEPROP.

3. Verify the changes were merged by reviewing the `openedge.properties` file in the *instance-name*`/conf` directory. The following properties (among others) were updated if you used the tutorial files:

```
[myProdInstance.ROOT.APSV]
    adapterEnabled=1

[AppServer.SessMgr.myProdInstance]
    agentStartupParam=-T "${catalina.base}/temp" pf
C:\OpenEdge\WRK\classicAppServerCode\databases.pf

[AppServer.Agent.myProdInstance]

PROPATH=${CATALINA_BASE}/openedge,${CATALINA_BASE}/openedge/events,${CATALINA_BASE}/webapps/ROOT/WEB-INF/openedge,[.
 . .]
    sessionShutdownProc=session_shutdown.p
    sessionStartupProc=session_startup.p
```

Move classic AppServer code to PAS for OpenEdge

After merging your classic AppServer application properties to your PAS for OpenEdge instance, you are ready to move your application code. Follow these steps:

1. Move your server code (not all connection types may be required for your application):
   a. Move ABL server code.
   b. Move WebSpeed server code.
   c. Move REST server code.
   d. Move SOAP server code.
2. Test your database access.
3. Start your PAS for OpenEdge instance.

Depending on your code base, you will need to complete some or all of the applicable sections to move your code to the proper transport. This tutorial demonstrates ABL client and application server code changes.

You can run your existing classic AppServer code from the current location, or you can take this opportunity to organize that code in the recommended application code and common code directories.

1. Move your server code

Move existing server code to these recommended locations to promote security, scalability, extensibility, and easier deployment:

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

1a. Move ABL server code

Move your ABL server code to a location on your new PAS for OpenEdge instance. This example uses the sample code from the `classicAppServerCode` directory, but all the ABL server code can be accessed similarly.

This tutorial runs sample code using the ROOT (default) web application on a PAS for OpenEdge instance. You will move the server code to the *instance-name*`/openedge` directory.

1. In the `%WRKDIR%/classicAppServerCode` directory, copy your ABL server code (`ServerGetCustNameSample.p`) to the *instance-name*`/openedge` directory. For example:

```
proenv>cd %WRKDIR%\classicAppServerCode
proenv>copy ServerGetCustNameSample.p %WRKDIR%\myProdInstance\openedge\
```

Note: Review the `ServerGetCustNameSample.p` to ensure it references the same database name that your server will connect to. For OpenEdge 12, the database is `sports2020`. For OpenEdge 11.7, the database is `sports2000`.

2. Verify that your *instance-name*`/conf/openedge.properties` includes the necessary entries in the `agentStartupParam` properties to reference your database.

   - If you are referencing the database directly, then ensure that the `agentStartupParam` database connection string points to the correct location of the database. For example:

```
[AppServer.SessMgr.myProdInstance]
 agentStartupParam=-T "C:\OpenEdge\WRK\myProdInstance/temp" -db
C:\OpenEdge\WRK\database\sports2020.db
```

   - If you are storing the database location in a `.pf` file, then ensure that the `.pf` file points to the correct location of the database. For example:

```
proenv>oeprop.{sh|bat} AppServer.SessMgr.myProdInstance.agentStartupParam
-T "C:\OpenEdge\WRK\myProdInstance/temp" -pf
C:\OpenEdge\WRK\classicAppServerCode\databases.pf
proenv>more %WRKDIR%\classicAppServerCode\databases.pf
 -db C:\OpenEdge\WRK\database\sports2020.db -H localhost
```

Note: Ensure you have updated the database connections in the `oemerge` file (where required) as described in Prepare to move from Classic AppServer to PAS for OpenEdge, including copying the Sports2020 database to the `%WRKDIR%\database` directory using `procopy`.

Note: If you are moving a classic **state-reset** or **state-aware** application, you must configure special **connect** and **disconnect** event procedures. For more information, see Migrate classic AppServer operating modes on page 38.

1b. Move WebSpeed server code

Note: This step is not required if you are using the sample tutorial code. It is included as a reference for moving WebSpeed application code to PAS for OpenEdge.

1. Move the static files (images and HTML pages), from your WebSpeed application to the *instance-name*/webapps/ROOT/static directory.
2. Review the PROPATH entry in the *instance-name*/conf/openedge.properties file to make sure that it includes compiled code for your WebSpeed application.
3. In the openedge.properties file, set the **defaultHandler** property to OpenEdge.Web.CompatibilityHandler. For example:

```
[myProdInstance.ROOT.WEB]
 defaultHandler=OpenEdge.Web.CompatibilityHandler
```

4. If you modified your web-disp.p, you must make similar changes to the default web-handler.p.

Note: OpenEdge does not support applications with HTML Mapped Web Objects.

For more information about WebSpeed code, see Migrate Classic WebSpeed Applications on page 77.

1c. Move REST server code

Note: This step can be performed using the sample tutorial code, but is not required if are not migrating classic AppServer REST services to PAS for OpenEdge. It is included as a reference for moving REST application code to PAS for OpenEdge.

A classic AppServer REST application .war or .zip file cannot be deployed to PAS for OpenEdge. If you have a classic AppServer REST application .war, then unzip the archive and deploy the application .paar file, or export the file from Progress Developer Studio for OpenEdge.

1. Copy the code that supports the REST interface API into the directory: *instance-name*/webapps/ROOT/WEB-INF/openedge. For example:

```
proenv>cd %WRKDIR%\classicAppServerCode\classicREST
proenv>copy getCustomer.p %WRKDIR%\myProdInstance\ROOT\WEB-INF\openedge
proenv>copy getAllCustomers.p %WRKDIR%\myProdInstance\ROOT\WEB-INF\openedge
```

Note: The sample tutorial files support the following REST service resources:

- /classicRESTService/customers
- /classicRESTService/customers/{custId}

2. Navigate to *instance-name*/bin. For example:

```
proenv>cd %WRKDIR%\myProdInstance\bin
```

3. Run the deployREST command with the path to the .paar file and the name of the ABL web application. For example:

```
proenv>deployREST %WRKDIR%\classicAppServerCode\classicREST\classicRESTService.paar
 ROOT
```

4. Update database connections as necessary in the `agentStartupParam` in the *instance-name*`/conf/openedge.properties` file, and ensure that database is running.
5. In the `openedge.properties` file, enable the REST transport for the ABL web application that supports the REST service. For example, if the ROOT application supports the REST service, set the following property:

```
[myProdInstance.ROOT.REST]
    adapterEnabled=1
```

6. When the PAS for OpenEdge is running, test the REST service connections with the following URLs:

```
http://hostname:port/rest/classicRESTService/customers
http://hostname:port/rest/classicRESTService/customers/3005
```

Note: The `3005` is a customer ID in the `Sports2020` database that is included with OpenEdge 12. If you are using the `Sports2000` database, use `1` as the customer ID. Additionally, if you are deploying to an ABL web application that is different from `ROOT`, include the ABL web application name in the URL before the `/rest` segment. For example:
`http://localhost:8817/ablwebapp_name/rest/classicRESTService/customers`

The deployREST command supports additional REST service descriptors other than `.paar` files, including `.zip` files that contain mobile catalog files (or other static files).

Note: **Do not** deploy the REST Manager (`oerm.war`) with PAS for OpenEdge because the supporting files are already included in the server.

For more information about deployREST, see Deploy REST services.

1d. Move SOAP server code

Note: This step is not required if you are using the sample tutorial code. It is included as a reference for moving SOAP application code to PAS for OpenEdge.

The Web Service Adapters (WSA) are not required for SOAP clients. PAS for OpenEdge includes the necessary SOAP transport support.

1. Copy the code that supports the SOAP interface API into one of the PROPATH locations.
2. Go to *instance-name*`/bin`.
3. Run `deploySOAP.{sh|bat} source_descriptor ROOT`

   Where:

`source_descriptor`	Specify the path of the source descriptor, which is a WSM file.
`service_name`	Specify the target service name.

For more information about deploySOAP, see Deploy SOAP services.

2. Test your database access

Before you start your PAS for OpenEdge instance, ensure the database server is running.

1. Using Proenv, start the database server using the `proserve` command, if it isn't already running. For example:

```
proserve %WRKDIR%\database\sports2020
```

   The database has successfully started when you see the `Multi-user session begin` and `Login by` messages.

2. Confirm that the `agentStartupParam` property in your *instance-name*`/conf/openedge.properties` file correctly references your database server.

   - If your `openedge.properties` references the database connection string directly, ensure you have the correct absolute path. For example:

```
[AppServer.SessMgr.myProdInstance]
 agentStartupParam=-T "C:\OpenEdge\WRK\myProdInstance/temp" -db
C:\OpenEdge\WRK\database\sports2020.db
```

   - If your `openedge.properties` references the database connection string through a `.pf` file, ensure the `.pf` file contains the correct absolute path. For example:

```
proenv>oeprop.{sh|bat} AppServer.SessMgr.myProdInstance.agentStartupParam
-T "C:\OpenEdge\WRK\myProdInstance/temp" -pf
C:\OpenEdge\WRK\classicAppServerCode\databases.pf
proenv>more %WRKDIR%\classicAppServerCode\databases.pf
 -db C:\OpenEdge\WRK\database\sports2020.db -H localhost
```

3. Start your PAS for OpenEdge instance

Before you test client connections, test the server startup procedure and database connectivity. In this step, start or restart the server to confirm that the startup procedures are available and the `.pf` file connections are working.

1. To start the instance, run **pasman pasoestart**. This loads your new `openedge.properties` file which enables the APSV transport on your production instance, a setting which is disabled by default on a new production instance.

```
pasman.{sh|bat} pasoestart -restart -I myProdInstance
```

   Where:

`-restart`	If the instance is already running or is in a hung state, attempt to stop it, and then execute a full start up. If the instance is already in the stopped state, this option has no effect.
`-I`	Specifies the instance name.

```
$pasman pasoestart -restart -I myProdInstance
Starting stopped PASOE instance myProdInstance
........

Start action: start
Initial state: stopped
Initial processes: 0
Exit state: started
Exit description: Starting stopped PASOE instance myProdInstance
Exit processes: 108182 108328
Exit status: 0
Exit errors:
$
```

2. View the application agent log file in a text editor to ensure that the agent started correctly and to see if your startup procedure ran.

   The agent log file can be viewed in *instance-name*/logs/*application-name*.agent.*date*.log. The agent log file should contain these types of entries indicate that agent startup, database connection, and execution of the startup procedure were successful:

```
2019-10-04T14:09:54.939-0400 027876 010356 1 AS-Aux-0 mtapsv:-:? MSAS Agent Starting
 Up -- Progress OpenEdge Release 12.1  build 1135.

2019-10-04T14:09:55.034-0400 027876 018972 2 AS-ResourceMgr mtapsv:-:? CONN
   Connected to database sports2020, user number 5. (9543)

2019-10-04T14:09:55.145-0400 027876 024664 1 AS-4 ?:?:? -- (Procedure:
'session_startup.p' Line:24) This is a session startup event procedure!
```

Troubleshooting tips:

- Review logs in the *instance-name*/logs directory for startup errors. The *instance-name*.agent.*{date}*.log file provides information about application agent connections, including database connectivity notifications.
- Recheck the recommendations in the openedge.properties file that were imported from the oemerge file.
- Confirm that the .pf file has the correct path to the database.
- Confirm that the event procedures are available in the PROPATH.

Connect clients with new PAS for OpenEdge transports

To connect client applications with new transports:

1. Update your client connections.
2. Test your client connections.
3. Stop your instance.

1. Update your client connections

All client connections must be updated to use one of the four transports, described in Welcome to PAS for OpenEdge. This table summarizes those updates:

Application changes by client types…	
OpenEdge	

Change clients of the AppServer Internet Adapter (AIA) to use the APSV transport	From:
	`-URL http://host:port/aia/Aia?AppService=brokername`
	To:
	`-URL http://host:port/apsv`

Change the ABL client connection to use the APSV transport	From:
	`-S port -H host -AppService brokername`
	To:
	`-URL http://host:port/apsv`

REST	

Change the client URLs to use the ROOT web application	From:
	`http://host:port/restservice/rest/restresource`
	To:
	`http://host:port/rest/restresource`

WSA	

Change the clients of the Web Services Adapter (WSA) to use the SOAP transport	From:
	`-WSDL http://host:port/wsa/wsa1/wsdl?targetURI=urn:CustomerSvc`
	To:
	`-WSDL http://host:port/soap/wsdl?targetURI=urn:CustomerSvc`

Application changes by client types…	
WebSpeed	

Change the WebSpeed clients to use the WEB transport	From:
	`http://host:port/cgi/wspd_cgi.sh/ . . .`
	To:
	`http://host:port/web/ . . .`
	*WSASP, WSISA, NSAPI, and CGIIP messengers are not used with PAS for OpenEdge.

To update an OpenEdge ABL client connection, follow these steps:

1. Open the `ClassicClient.p` file in an editor. For example:

```
proenv>mpro C:\OpenEdge\WRK\classicAppServerCode\ClassicClient.p
```

2. Inspect the `ClassicClient.p` file, specifically the `cConnect` string:

```
ASSIGN iCustNum=3005.

CREATE SERVER hServer.

cConnect = "-S 5162 -H localhost -AppService asbroker1".

lReturn= hServer:CONNECT(cConnect).

IF (lReturn) THEN DO:
   RUN ServerGetCustNameSample.p ON hServer (INPUT iCustNum, OUTPUT cCustName).
   DISPLAY "Customer Name: " cCustName FORMAT "x(40)" SKIP.
   hServer:DISCONNECT().
END.

OUTPUT CLOSE.

QUIT.
```

Note: If you are using OpenEdge 11.7, then change the assignment statement from `iCustNum=3005` to `iCustNum=10` in the `ClassicClient.p` file.

3. Replace the classic connections (`cConnect`) parameter with the updated connection parameter, and indicate that the `sessionModel` is `Session-Free`. For example:

```
ASSIGN iCustNum=3005.

CREATE SERVER hServer.

cConnect = "-URL http://localhost:8817/apsv -sessionModel Session-Free".

lReturn= hServer:CONNECT(cConnect).
```

```
IF (lReturn) THEN DO:
   RUN ServerGetCustNameSample.p ON hServer (INPUT iCustNum, OUTPUT cCustName).
   DISPLAY "Customer Name: " cCustName FORMAT "x(40)" SKIP.
   hServer:DISCONNECT().
END.

OUTPUT CLOSE.

QUIT.
```

4. Save the file as `PASforOpenEdgeClient.p`.
5. Recompile the client code and exit the file.
6. Open the server code file—`ServerGetCustNameSample.p`—in an editor. The code takes a customer number as input and returns a customer name:

```
/* Sample code, run on Appserver, gets customer's name
   from "sports2020" database, based on customer number
   sent from client. */

DEF INPUT PARAM iCustNum AS INTEGER.
DEF OUTPUT PARAM cCustName AS CHAR.

IF CONNECTED("sports2020") THEN DO:
      FIND FIRST customer WHERE custNum = iCustNum NO-LOCK NO-ERROR.
      IF AVAILABLE customer THEN
          cCustName = Name.
      ELSE
          cCustName = "No record".
```

Note: No changes are required on the server code, `ServerGetCustNameSample.p`, if you are running OpenEdge 12. If you are using OpenEdge 11.7, then change the database name from `sports2020` to `sports2000` in the `ServerGetCustNameSample.p` file.

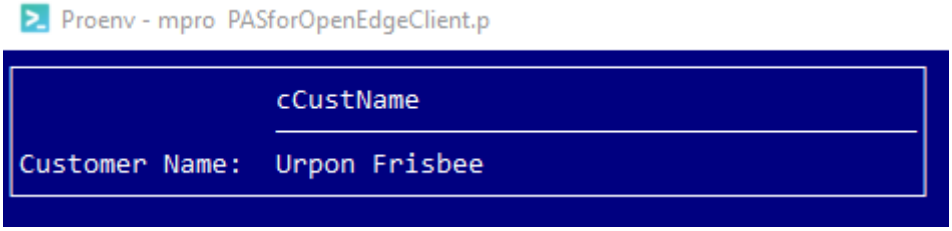
Test your client connections

The final test is to run a client connection that collects data from your database.

1. Run the client code. For example, open the `PASforOpenEdgeClient.p` file in mpro using the following command in Proenv and then press F1:

```
proenv>mpro %WRKDIR%\classicAppServerCode\PASforOpenEdgeClient.p
```

2. Verify that your application can connect and access your database data:



Stop your instance

When you are done testing your server and it is no longer needed, then it is a good practice to stop the instance.

1. Use the `pasman` command with the `stop` action to stop a running instance.

```
pasman.{sh|bat} stop -I myProdInstance
```

2. View the application agent log file in a text editor to ensure that the agent stopped correctly and to see if your shutdown procedure ran. If you are using the files included in this tutorial, then the shutdown procedure writes the following message to the log file:

```
This is a session shutdown event procedure!
```

The agent log file can be viewed in *instance-name*`/logs/`*application-name*`.agent.`*date*`.log`. The agent log file should contain these types of entries that indicate a successful execution of the shutdown procedure, database disconnection, and agent shutdown:

```
2019-10-04T14:36:42.252-0400 027876 019344 1 AS-4 ROOT:a:0000001f -- (Procedure:
'session_shutdown.p' Line:22) This is a session shutdown event procedure!

2019-10-04T14:36:42.276-0400 027876 018972 2 AS-ResourceMgr mtapsv:-:? CONN
   Disconnecting from database sports2020, user number 5. (9545)

2019-10-04T14:36:42.279-0400 027876 010356 1 AS-Listener mtapsv:-:? MSAS Agent Shutdown
 Complete.
```

To prepare the test instance for full production release, continue your learning with the following:

- Important details about your new server
- Next steps with PAS for OpenEdge on page 34

Important details about your new server

Here are a few important details about your new server:

PAS for OpenEdge differs fundamentally from the classic AppServer

Although both PAS for OpenEdge and the classic AppServer run ABL business applications, the architecture and configuration are fundamentally different. PAS for OpenEdge is an application server based on a web server that uses special web applications to run ABL code. It has all the behavior and features of a web server. Although many of the configuration parameters appear to be similar, the tools and techniques for a successful deployment in the multi-session environment are different. For more information about configuring your instance, see the output of the property conversion tool (*instance-name*`/conf/`*abl_app_name*`.`*ubroker_name*`.oemerge`).

The latest release keeps you up to date with security and software updates

In the web environment, **tracking the latest security changes is extremely important**. The PAS for OpenEdge product is continually being updated with new functionality, bug-fixes, and security patches. Stay current to keep your application secure and stable.

Use property files to change PAS for OpenEdge configuration values

Although PAS for OpenEdge is based on Apache Tomcat, PAS for OpenEdge has simplified the underlying Tomcat configuration and startup for local and remote administration. Rather than making changes to the *instance-name*/conf/server.xml or *instance-name*/bin/setenv.[bat|sh] files directly, PAS for OpenEdge uses **property files** and customer extensible script files. For more information, see Property files.

Check all PAS for OpenEdge log files for errors

A client request traverses multiple PAS for OpenEdge subsystems, any one of which can raise an execution error. The recording of the error occurs in the subsystem-specific log file. Always check all of the log files in the *instance-name*/logs directory when you investigate execution errors. For more information, see Log files.

PAS for OpenEdge may fail to start because of errors in your ABL application

The PAS for OpenEdge multi-session agent log file distinguishes between ERROR, WARNING, or INFO messages. As a result, the PAS for OpenEdge pasoestart command can report only whether the agent OS process stopped due to startup problems. If pasoestart does not specifically report the startup error, then manually inspect the agent log file in the *instance-name*/logs directory (*abl_app_name*.agent.*{date}*.log) to determine what the startup problem is and correct it.

Configure and load test your instance before moving into production

Choosing the right machine image size and optimal PAS for OpenEdge configuration for that machine image is key in moving your ABL application into the production environment. PAS for OpenEdge is optimized for resource consumption and throughput under load. Unlike in classic AppServer or WebSpeed, you cannot derive ABL application performance using a single client. Test your application with the anticipated number of concurrent clients to ensure that you do not exceed finite resource limits imposed by the OS, OS processes, networking, and OpenEdge database. For more information, see Goals and common steps for tuning PAS for OpenEdge instances.

Stopping a PAS for OpenEdge instance may take some time

Stopping a PAS for OpenEdge instance may take several minutes in some cases. It may even appear hung. PAS for OpenEdge (Tomcat) has a policy that allows its web applications to finish client requests before stopping. A normal PAS for OpenEdge stop does not always stop all of its processes. If you want PAS for OpenEdge not to wait for client requests to finish before stopping, use the pasman stop command with the -F option to force the shutdown.

The Apache Tomcat access log is a great troubleshooting tool

The access log is a great help when you troubleshoot client connection problems, maximum response times, client login failures from hackers, and much more. Even though the access log uses a small amount of processing time and disk space, you cannot get this information easily from other sources and it can be helpful in monitoring the health of your server and tuning its run time. See the Apache Tomcat® Configuration Reference.

Apache Tomcat stuck thread detection is a useful alerting mechanism

The Apache Tomcat stuck thread detection identifies requests that take a long time to process, which may indicate that something is wrong in the request execution. Stuck thread detection is an alerting mechanism that logs a message for requests that take longer than a configurable amount of time to complete. See the Apache Tomcat® Configuration Reference.

UNIX file permissions that only allow the root use

On UNIX, the PAS for OpenEdge production installation and instance tailoring eliminates all access by any user account. After an OpenEdge installation completes, its instances are configured to be started, stopped, configured, and monitored by the root user account (required by the OpenEdge installer). If all your OS administration is performed via the root user account, no action needs to take place. If you follow best practices and never use the OS root user account, then you will need to change file permissions for core and instance executables and scripts. See About user and file permissions.

Next steps with PAS for OpenEdge

If you are …

- Interested in seeing a demonstration, watch and share the video: Moving your classic AppServer applications to the Progress® Application Server for OpenEdge®
- A system administration supporting the new server, take the online training: Progress Application Server for OpenEdge Administration (1000-095)
- Writing new REST applications for PAS for OpenEdge, take advantage of the ABL-based WebHandler interfaces for Web application development (like Java servlet APIs): Develop an ABL service using the WEB transport
- Migrating a classic state-reset or state-aware application, configure special connect and disconnect event procedures: Migrate classic AppServer operating modes
- Migrating SSL server certificates: Digital certificate management
- Tuning PAS for OpenEdge for performance: Tune PAS for OpenEdge instances
- Monitoring PAS for OpenEdge: Monitor PAS for OpenEdge instances
- Registering, unregistering, or deleting PAS for OpenEdge instances: Create and configure PAS for OpenEdge instances.
- Registering a PAS for OpenEdge instance as a Windows service: Register a PAS for OpenEdge instance as a Windows service.
- Managing PAS for OpenEdge with OpenEdge Management: Learn about PAS for OpenEdge and OpenEdge Management.

3

ABL Application Code Migration

This content describes the concepts needed for migrating ABL code to PAS for OpenEdge, including:

- Best practices for ABL application structure
- Differences between the classic AppServer and PAS for OpenEdge
- Migrating classic AppServer client connections to PAS for OpenEdge

For details, see the following topics:

- ABL application structure
- Migrate classic AppServer operating modes
- Differences in event procedures between classic AppServer and PAS for OpenEdge
- Database connections in PAS for OpenEdge
- ABL changes in PAS for OpenEdge
- Environment settings in PAS for OpenEdge
- File system directories and paths in PAS for OpenEdge
- Migrate client connections to PAS for OpenEdge

ABL application structure

Migrating an application to PAS for OpenEdge means reorganizing your application structure with the goal of making applications easier to deploy, scale, and extend. PAS for OpenEdge is a web server, and your applications will move toward a web server application structure. Code can be organized at the following levels:

- **Install** directories
- **Instance** directories
- **Application** directories
- **Service** directories

Install directories

The PAS for OpenEdge **install** directory enables you to have a default behavior across all instances. Having this behavior separated into the installation directory structure makes it easier to upgrade to a new version of the PAS for OpenEdge server without losing the customizations that were made to the individual instances.

Instance directories

The **instance** directories enable you to customize each instance. Changes made at the instance level override those at the installation level. Each instance contains the following structures and related content:

- **`bin`**—OpenEdge and custom utilities and libraries that are tailored to that specific instance. For example, OpenEdge utilities are tailored to use the file space of the local instance but load and use installed shared libraries and utilities.
- **`common/lib`**—Java libraries and files that are shared across all of the web applications in the instance.

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

- **`conf`**—The Tomcat, OpenEdge, and custom configuration files for the instance that control only the applications deployed into the instance.
- **`logs`**—The log files for the server instance, including all of the server, web application, OpenEdge, and ABL application log files.
- **`openedge`**—The `.p` and `.r` files and supporting files only for ABL applications. Other Progress and third-party web applications cannot write or read from this directory.
- **`temp`**—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-written ABL web applications for the instance.
- **`work`** —Application-generated persistent working files.

  Note: These files should not be deleted by an administrator as part of cleaning up before restarting an application.

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

Application directories

A typical **ABL web application** directory structure may follow the model set by the OpenEdge ABL web application (`oeabl.war`). Copies of `oeabl.war` can be deployed to your instance from the `$DLC/servers/pasoe/extras` directory.

- **`/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).
- **`/web`**—A URI path reserved for classic WebSpeed applications and new REST clients (can be disabled).
- **`static`**—A directory that contains static files, including images, style sheets, and HTML pages.
- **`META-INF`**—A required directory that contains 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 web application's configuration file (`web.xml`), Spring Security configuration files, and other application private files.
  - **`adapters`**—OpenEdge directory tree that contains the REST and SOAP deployment files.
  - **`lib`**—Required directory that contains any Java libraries specific to the web application.
  - **`classes`**—Required directory that contains any Java classes or data files specific to the web application.
  - **`openedge`**—Directory provided by OpenEdge where ABL developers can distribute their ABL application files that apply specifically to the web application

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

While an ABL application could physically locate their files anywhere in the OS's file system, locating them inside of the PAS for OpenEdge web application instances promotes scaling, extensibility, and deployment. By organizing files, it is easier to:

- Deploy an ABL application (identified by name) to an existing PAS for OpenEdge instance.
- Secure an application by configuring it as read-only. For example, common `.r` and `.p` files used across multiple web applications located in the `openedge/*` directories of the instance.
- 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, REST, and WEB transport access to ABL business logic.

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

ABL service directories

An ABL service is a transport-dependent path into an ABL application's business logic (business entities). The ABL service names are defined at development time and do not change at deployment.

Migrate classic AppServer operating modes

In the classic OpenEdge AppServer, the operating modes for **session-managed** client connections is determined in the classic AppServer configuration. The possible operating modes for session managed connections are **state-reset**, **state-aware**, and **stateless**. For **session-free** client connections, the operating mode is always **state-free**. Each classic AppServer instance only runs in only one model. You had to start multiple classic AppServers in order to support multiple session models and operating modes.

The three operating modes in the classic AppServer produced different requirements for implementing session-managed client connections. In some session-managed modes, the session's client request context is preserved between client requests because the client was **bound** to a single ABL session. Other session-managed models result in the ABL application manually managing client request context in an external store because each request executed in a different ABL session. When a client uses the session-free model, the ABL application manually manages the client request context in an external store because all client requests execute in a different ABL session.

In PAS for OpenEdge, the client continues to connect using the session-managed or session-free models, which results in the same client-side behavior as with the classic AppServer. However, the session model is simplified in PAS for OpenEdge, since now there is only one supported connection model, **HTTP/S**. By removing all other connection models, the session models (now called **application models**) become simply session-managed or session-free, where any PAS for OpenEdge instance is capable of supporting both types simultaneously.

A PAS for OpenEdge session-managed connection is **conditionally bound** to the same ABL session based on the setting of the `SESSION:SERVER-CONNECTION-BOUND-REQUEST` attribute. Depending on this setting, the optimizations of the multi-session agent provide you with the resource scaling of the classic AppServer's stateless operating mode with the performance and simplicity of session context management of the state-reset and state-aware operating modes. This provides you with the best of both worlds. When the ABL application requires maximum scalability at the request level, or actual concurrent execution of client requests, the PAS for OpenEdge session-free model operates with the same behavior as the classic AppServer running in the state-free operating mode.

With PAS for OpenEdge merging all of the classic AppServer's session-managed modes, there are some differences between when a PAS for OpenEdge session's event procedures execute compared with the corresponding classic AppServer's configuration procedures. Most changes are transparent. However, in some cases, you must make adjustments if you want your ABL sessions on the server to execute event procedures with the same behavior as configuration procedures on the AppServer.

The following tables compare configuration procedure execution in the classic AppServer and event procedure execution in PAS for OpenEdge:

Table 1: Session-managed event procedures

Procedure	Classic AppServer	PAS for OpenEdge	Comments
Startup	Always	Always	No change ; run persistent
Shutdown	Always	Always	No change
Connect	Always	Always	No change ; run persistent
Disconnect	Always	Always	No change

Procedure	Classic AppServer	PAS for OpenEdge	Comments
Activate	Never(1) ; Conditional(2)	Always	PAS for OpenEdge follows classic stateless behavior, as long as `SESSION:SERVER-CONNECTION-BOUND = FALSE`
Deactivate	Never(1) ; Conditional(2)	Always	PAS for OpenEdge follows classic stateless behavior, as long as `SESSION:SERVER-CONNECTION-BOUND = FALSE`

Table 2: Session-free event procedures

Procedure	Classic AppServer	PAS for OpenEdge	Comments
Startup	Always	Always	No change
Shutdown	Always	Always	No change
Connect	Never	Never	No change
Disconnect	Never	Never	No change
Activate	Conditional(3)	Always	PAS for OpenEdge follows classic state-free behavior, as long as `SESSION:SERVER-CONNECTION-BOUND = FALSE`
Deactivate	Conditional(3)	Always	PAS for OpenEdge follows classic state-free behavior, as long as `SESSION:SERVER-CONNECTION-BOUND = FALSE`

1. Classic AppServer state-reset and state-aware operating modes never execute Activate or Deactivate event procedures.
2. Classic AppServer stateless operating mode always executes Activate and Deactivate procedures unless the client is in a bound state.
3. Classic AppServer state-free operating mode always executes Activate and Deactivate procedures unless the client is in a bound state.

When a PAS for OpenEdge client connects using the session-managed application model, by default, server sessions execute requests from this client in the unbound state, similar to the classic stateless operating mode. As on the classic AppServer, you can change the client connection to the bound state by having the client run a remote persistent procedure or by having the server session set the `SESSION:SERVER-CONNECTION-BOUND-REQUEST` attribute to `TRUE`. So, to emulate the behavior of the classic AppServer state-reset and state-aware operating modes, you must set this attribute appropriately in the Connect and Disconnect event procedures of the PAS for OpenEdge instance. In addition, to emulate the classic AppServer state-reset operating mode, you need to execute `QUIT` as the last statement of the Disconnect procedure.

When a PAS for OpenEdge client connects using the session-free application model, server sessions also execute requests from this client in the unbound state, similar to the classic state-free operating mode. As on the classic AppServer, you can change the client connection to the bound state by having the client run a remote persistent procedure, but also as on the classic AppServer, the session-managed `SESSION` handle attributes have no function. For example, setting the `SESSION:SERVER-CONNECTION-BOUND-REQUEST` attribute to `TRUE` in a server session running in the unbound state has no effect. You can only know that the connection is bound by querying the `SESSION:SERVER-CONNECTION-BOUND` attribute. And, as in the case of the classic AppServer, Progress recommends that you **never** have PAS for OpenEdge clients execute remote persistent procedures to make a session-free connection bound.

So, if you are migrating an application from a classic stateless or state-free AppServer, your application should require no ABL code changes to achieve the same behavior on PAS for OpenEdge.

The following topics detail any changes that might be required to match your particular classic AppServer behavior.

Migrate classic state-reset operating mode

To achieve the same client **bound** and **reset** behavior in PAS for OpenEdge as in a classic state-reset AppServer, include the following as the first statement in your Connect event procedure:

```
SESSION:SERVER-CONNECTION-BOUND-REQUEST = TRUE.
```

This code disables the execution of Activate and Deactivate event procedures for client requests over this connection until this attribute is set to `FALSE`, but leaves them available for use by session-free client connections.

Also include the following as the last statements before exiting your Disconnect event procedure:

```
SESSION:SERVER-CONNECTION-BOUND-REQUEST = FALSE.
QUIT.
```

The `QUIT` statement in PAS for OpenEdge effectively performs the same ABL session reset operation that occurs in the classic AppServer after the Disconnect event procedure executes.

Migrate classic state-aware operating mode

To achieve the same client **bound** behavior in PAS for OpenEdge as in a classic state-aware AppServer, include the following as the first statement in your Connect event procedure:

```
SESSION:SERVER-CONNECTION-BOUND-REQUEST = TRUE.
```

This disables the execution of Activate and Deactivate event procedures for client requests over this connection until this attribute is set to FALSE, but leaves them available for use by session-free client connections.

Also include the following as the last statement before exiting your Disconnect procedure:

```
SESSION:SERVER-CONNECTION-BOUND-REQUEST = FALSE.
```

Migrate classic stateless operating mode

No changes are required to event procedures to migrate applications running on a classic stateless AppServer.

Migrate classic state-free operating mode

No changes are required to event procedures to migrate applications running on a classic state-free AppServer.

Differences in event procedures between classic AppServer and PAS for OpenEdge

PAS for OpenEdge uses a multi-session agent, which means that a single agent can concurrently handle requests from several clients, each in its own session, and each running its own application model. This way of executing ABL applications could change your application's use of the event procedures.

PAS for OpenEdge has only two operating modes (application models): session-managed and session-free. The classic AppServer operating mode of stateless most closely matches session-managed. State-free matches session-free. If your application runs in either state-reset or state-aware operating modes on the classic AppServer, then you may need to modify your Connect and Disconnect procedures to mimic these modes when running on PAS for OpenEdge. For more information, see Migrate classic AppServer operating modes on page 38.

Multi-session agent Startup and Shutdown procedures

The PAS for OpenEdge multi-session agent has two additional Startup and Shutdown event procedures that run when the agent starts up and shuts down. The following table describes the behavior of these two procedures. The **Procedure name** column lists the property in `openedge.properties` where you set the name of the specified procedure:

Table 3: Multi-session Agent Startup and Shutdown procedures

Procedure name	Persistence	Runs at	Parameters
`agentStartupProc`	Non-persistent	Multi-session agent process startup	A single `CHARACTER` argument passed to the procedure as an `INPUT` parameter and specified as the `agentStartupProcParam` property in `openedge.properties`
`agentShutdownProc`	Non-persistent	Multi-session agent process shutdown	None

Note: The `agentStartupProc` procedure runs in a separate ABL session within the multi-session agent. Any global values it creates are not visible to other ABL sessions.

Session Startup and Shutdown procedures

The classic AppServer provides ABL session Startup and Shutdown event procedures that allow you to manage the initialization and cleanup of an ABL session. The properties that control these event procedures use the prefix `srvr`, which reflects their identification with server (agent) processes of the classic AppServer. Each classic AppServer agent is a single ABL session that runs as a single OS process (AVM). These event procedures have always been executed as part of the ABL session startup and shutdown.

In PAS for OpenEdge, these Session Startup and Shutdown procedure properties are renamed with the `session` prefix to better reflect their association with ABL sessions rather than server processes. In addition, PAS for OpenEdge eliminated inconsistent execution produced by connection models that no longer exist (HTTP is the only one). The multi-session agent in PAS for OpenEdge starts ABL sessions on demand to meet client load, and multi-session agent stops ABL sessions when they are idle for a certain period of time.

Naming

The classic AppServer properties in `ubroker.properties` are `srvrStartupProc`, `srvrStartupParam`, and `srvrShutdownProc`.

The corresponding PAS for OpenEdge properties in `openedge.properties` are `sessionStartupProc`, `sessionStartupProcParam`, and `sessionShutdownProc`.

In both servers, these properties are implemented as *`parameter-name`*`=`*`value`* pairs.

Execution

The following table compares the execution of these procedures on the classic AppServer and PAS for OpenEdge. The **Procedure name** column lists the property in `openedge.properties` where you set the name of the specified procedure:

Table 4: Comparing Session Startup and Shutdown procedures between AppServer and PAS for OpenEdge

Procedure name	AppServer persistence	PAS for OpenEdge persistence	Runs at	Startup Parameters
sessionStartupProc	Persistent	Persistent	ABL session startup	A string of session startup parameters specified as the sessionStartupProcParam property in openedge.properties
sessionShutdownProc	Non-persistent	Non-persistent	ABL session shutdown	None

In all other respects, the Session Startup and Shutdown event procedures are the same for classic AppServer and PAS for OpenEdge.

Connect and Disconnect procedures

The classic AppServer provides ABL session Connect and Disconnect event procedures that enable you to manage the initialization and cleanup of a session-managed server session. Because of the different connection models for the classic AppServer, the execution and scope of these procedures vary on the classic AppServer. The properties that control these event procedures use the prefix `srvr`, which reflects their identification with server (agent) processes. Each classic AppServer agent is a single ABL session that runs as a single OS process (AVM).

In PAS for OpenEdge, these Connect and Disconnect procedure properties are renamed with the `session` prefix to better reflect their association with ABL sessions rather than server processes. In addition, PAS for OpenEdge eliminated inconsistent execution produced by connection models that no longer exist (HTTP is the only one).

Naming

The classic AppServer properties in `ubroker.properties` are `srvrConnectProc` and `srvrDisconnProc`.

The corresponding PAS for OpenEdge properties in `openedge.properties` are `sessionConnectProc` and `sessionDisconnProc`.

In both servers, these properties are implemented as *`parameter-name=value`* pairs.

This `sessionDisconnProc` procedure is important when you migrate a state-reset application to PAS for OpenEdge. Because there is no "reset" of session-managed sessions in PAS for OpenEdge, you must write additional code to reset the session and remove any context information before the next client uses the session.

Execution

The following table compares the execution of these procedures on the classic AppServer and PAS for OpenEdge. The **Procedure name** column lists the property in `openedge.properties` where you set the name of the specified procedure:

Table 5: Comparing Connect and Disconnect procedures between AppServer and PAS for OpenEdge

Procedure name	AppServer persistence	PAS for OpenEdge persistence	Runs at	Parameters
`sessionConnectProc`	Persistent	Persistent	OpenEdge client connection	No changes
`sessionDisconnProc`	Non-persistent	Non-persistent	OpenEdge client disconnect and session shutdown	None

Notes:

- The classic AppServer's `srvrConnectProc` persistent procedure is not deleted unless the ABL application code explicitly deletes it. PAS for OpenEdge automatically deletes the `sessionConnectProc` procedure after the client connection is closed.
- The classic stateless AppServer `srvrConnectProc` and `srvrDisconnectProc` procedures execute in different ABL sessions. Because PAS for OpenEdge creates a bound, session-managed client connection to the same ABL session, the `sessionConnectProc` and `sessionDisconnectProc` event procedures execute in the same ABL session as all of the client's requests (the same way the classic state-reset and state-aware AppServer configuration procedures work).
- The classic state-reset AppServer affects the automatic "reset" of the ABL session after client disconnect. To enable PAS for OpenEdge to emulate this classic state-reset behavior, with no operating modes, execute an ABL `QUIT` statement at the end of the `sessionDisconnProc` after setting `SESSION:CLIENT-SERVER-BOUND-REQUEST` to `FALSE`.
- A classic state-reset or state-aware AppServer affects the automatic client-bound state before the `srvrConnectProc` procedure is executed. To allow PAS for OpenEdge to emulate this behavior, with no operating modes, set the ABL `SESSION:CLIENT-SERVER-BOUND-REQUEST` attribute to `TRUE` at the beginning of the `sessionConnectProc` procedure to produce the same behavior. (See the `sessionActivateProc` and `sessionDeactivateProc` properties in Activate and Deactivate procedures on page 44.)

In all other respects, the ABL session's Connect and Disconnect event procedures are the same for classic AppServer and PAS for OpenEdge.

Activate and Deactivate procedures

The classic AppServer provides ABL session Activate and Deactivate event procedures that enable you to manage the initialization and cleanup of a client's request in both unbound session-managed and session-free models. Because of the different connection models for the classic AppServer, the execution and scope of these procedures vary in the AppServer, based on limitations imposed by the an application's architecture. The properties that control these event procedures use the prefix `srvr`, which reflects their identification with server (agent) processes. Each AppServer agent is a single ABL session that runs as a single OS process (AVM).

The `srvrActivateProc` and `srvrDeactivateProc` procedures are important if your application is using the classic AppServer's stateless or state-free modes. Those classic AppServer modes require that you manually manage client and session context because each request is executed in a different ABL session. The Activate procedure is run because it restores the context saved at the end of the client's previous request, and the Deactivate procedures is run because it saves the context for use by the client's next request. Because there are no formal connection and disconnection operations in the classic AppServer modes, you must implement your own start/stop client session architecture.

The classic AppServer executes the Activate and Deactivate procedures. The client bound condition relates to the AppServer client being bound to a single ABL session where each client request is guaranteed to execute in that particular ABL session. When the client is bound to a session, there is no requirement for the ABL application code to save and restore context using the Activate and Deactivate procedures because the context persists as long as the session is bound.

Whenever an ABL session is client bound, it will not execute Activate and Deactivate procedures. The classic AppServer's modes influence the client bound condition and the ABL code in the application. The following table helps to illustrate the classic AppServer's behavior:

Table 6: Comparing Activate and Deactivate procedures between AppServer and PAS for OpenEdge

Operating mode	Starting bound state	Ending bound state	Can change bound state?
State-reset	Bound	Bound	No
State-aware	Bound	Bound	No
Stateless	Unbound	Unbound	Yes
State-free	Unbound	Unbound	Yes

The stateless and state-free classic AppServer modes can change the state from unbound to bound, and from bound to unbound based on two conditions:

1. The client executes a persistent procedure on the AppServer (stateless or state-free).
   - The Activate procedure executes before the persistent procedure's main block.
   - The client bound state is set after the Activate procedure runs.
   - Because the client bound state exists, the Deactivate procedure does not execute after the persistent procedure's main block executes.
   - Activate or Deactivate procedures do not execute until the client bound state is cleared.
   - The client bound state is cleared when the persistent procedure is deleted by the client, and the Deactivate procedure is executed as part of the delete operation.

2. The classic AppServer ABL code sets `SESSION:SERVER-CONNECTION-BOUND-REQUEST` to `TRUE` (stateless only).
   - The Deactivate procedure does not execute at the end of the client request that set the client bound state.
   - Activate or Deactivate procedures do not execute until the client bound state is cleared by the ABL application code running in the classic AppServer.
   - When the AppServer's ABL code clears the client bound state, the Deactivate procedure will execute when the current procedure ends.
   - Normal Activate and Deactivate execution resumes on each subsequent client request after the client bound state is cleared.

Like the classic AppServer, PAS for OpenEdge supports the client bound state, but might require changes to the ABL code to get the same behavior as classic AppServer operating modes when a client is connected with a given application model:

Application model	Starting bound state	Ending bound state	Can change bound state?
Session-managed	Unbound	Unbound	Yes
Session-free	Unbound	Unbound	Yes

The transitions from bound to unbound and unbound to bound in PAS for OpenEdge follow the same behavior as the classic AppServer for client executed persistent procedures or using the `SESSION:SERVER-CONNECTION-BOUND-REQUEST` attribute on the server. Likewise, the state changing sequence of PAS for OpenEdge Activate and Deactivate procedures follow the behavior of the changing client bound states for the AppServer. The difference between classic AppServer and PAS for OpenEdge is that to start a server session in the bound state you must change the session-managed Connect event procedure to manually establish the client bound state. When you manually force a client bound state in PAS for OpenEdge, ensure that the client bound state is cleared in the session-managed Disconnect event procedure.

Naming

The classic OpenEdge AppServer properties in `ubroker.properties` are `srvrActivateProc` and `srvrDeactivateProc`.

The corresponding PAS for OpenEdge properties in `openedge.properties` are `sessionActivateProc` and `sessionDeactivateProc`.

In both servers, these properties are implemented as *`parameter-name`*=*`value`* pairs.

Database connections in PAS for OpenEdge

The OpenEdge database network connection policy did not change in PAS for OpenEdge. Like the classic AppServer, it is one private network connection per ABL session in the multi-session agent. Therefore, the application code and PROMON monitoring are the same for both servers.

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

Two new **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's connection

---

Note: Manually disconnecting `SELF/PASA` results in disconnecting **all** ABL session users that were using the shared self-service database connection, which causes all database connectivity to be dropped.

ABL changes in PAS for OpenEdge

ABL changes in PAS for OpenEdge include:

- Modifications to the existing ABL
- New ABL specifically for PAS for OpenEdge

Modifications to the existing ABL

While very little has changed in existing ABL support, the following table lists a few statements, handle attributes, and class properties whose behavior or values changed when they are executed or queried in a PAS for OpenEdge ABL session:

Table 7: AppServer ABL changed for PAS for OpenEdge

Statement, handle attribute, or class property	Behavior or value range on PAS for OpenEdge
`QUIT` statement	Clears the server session context before terminating the current client request and returning the session to the agent's session pool for access by other clients
`SESSION:SERVER-OPERATING-MODE` attribute	`"Session-Managed"` or `"Session-Free"`
`SESSION:CLIENT-TYPE` attribute	`"MULTI-SESSION-AGENT"`
`SESSION:LOCAL-VERSION-INFO:OEClientType` class property	`"MULTI-SESSION-AGENT"`

New ABL specifically for PAS for OpenEdge

The following table lists new ABL statement options and class properties that were added specifically for PAS for OpenEdge:

Table 8: New ABL added for PAS for OpenEdge

Statement option or class property	Value range	Description
`THREAD-SAFE` option of the `PROCEDURE` statement	N/A	Tells the PAS for OpenEdge session manager that the declared DLL or shared library procedure is thread safe so it can run in multiple server sessions simultaneously
`SESSION:CURRENT-REQUEST-INFO:AgentId` class property*	Positive integer	Returns the ID of the current multi-session agent
`SESSION:CURRENT-REQUEST-INFO:SessionId` class property*	Positive integer	Returns the ID of the current ABL session within the current multi-session agent

Statement option or class property	Value range	Description
SESSION:CURRENT-REQUEST-INFO:ThreadID class property*	Positive integer	Returns the ID of the AVM running the current ABL session within the current multi-session agent
SESSION:CURRENT-REQUEST-INFO:AdapterType class property*		Returns an enumerated value representing the client type that made the request.

Note: An asterisk (*) indicates that this ID is dynamic and can change each time a PAS for OpenEdge instance starts.

Environment settings in PAS for OpenEdge

Environment variables for the classic AppServer are set in the `$DLC/properties/ubroker.properties` file in a section similar to the following:

```
[Environment.asbroker1]
TESTENV=MYEN
```

In PAS for OpenEdge, you do not set environment variables in configuration files like `ubroker.properties` on the classic AppServer. Instead, you set environment variables using the following method:

- **At the application level** — Define a *myenv*`_setenv.{sh|bat}` script, and place it in the `instance-name/bin` directory.

  For more information, see "Modify environment variables" in *Manage Progress Application Server (PAS) for OpenEdge*.

Note: On UNIX platforms, your `myenv_setenv.sh` script does need not be marked executable. You must `EXPORT` any environment variables defined in that script so other processes can use them.

File system directories and paths in PAS for OpenEdge

PAS for OpenEdge runs within a web application server environment. For an ABL application to run consistently and securely within that environment, the default location of certain file system paths must be different than the classic AppServer. The following table shows a list of known differences:

Table 9: File system directories and paths in PAS for OpenEdge and classic AppServer

Path	Classic AppServer	PAS for OpenEdge	Comments
Working directory	`$WRKDIR`	`$CATALINA_BASE/work`	Relocated to the PAS for OpenEdge instance's `work` directory
Temporary directory	`CWD` (current working directory)	`$CATALINA_BASE/temp`	Relocated to PAS for OpenEdge instance's `temp` directory
`PROPATH`	`$WRKDIR`	`$CATALINA_BASE/openedge`	Relocated so that the ABL application's code is located in the instance that runs it

Migrate client connections to PAS for OpenEdge

When you migrate clients from accessing the classic AppServer or its adapters to accessing PAS for OpenEdge, search for the statements or routines that invoke the connection code and ensure that it uses an appropriate URL to connect to the ABL web application, including a transport, service, and resource path appropriate to the client type.

For ABL clients, this is the `CONNECT()` method on the server object handle used to access the PAS for OpenEdge instance. For OpenEdge Java and .NET Open Clients, you pass a set of parameters to the `Connection` object constructor. URLs for these clients use the APSV transport, but no service and resource path.

For OpenEdge SOAP web service clients, the WSDL file that you deploy provides the URL information for SOAP web service clients to access its service objects. Connection URLs for these clients use the SOAP transport followed by a WSDL path that includes a URL query parameter to specify the web service object to access.

OpenEdge REST web service clients connect with a URL that uses the REST transport followed by a URI path that identifies the service and resource. The format of the URL does not change from accessing the same REST web service implemented on the classic AppServer. You can thus configure a PAS for OpenEdge instance and ABL web application so that existing REST web service clients can use URLs that are identical to the URLs they used to access the same REST web services running on the classic AppServer.

In general, the URL that you use to access an ABL web application must conform to the following syntax:

Syntax

instance/[*oeabl-web-application/*]*transport*[*/service-resource*]

instance

Specifies the PAS for OpenEdge instance to connect with using the following syntax:

Syntax

```
scheme://[username:passphrase@]host:port
```

scheme

Specifies either HTTP or HTTPS.

Note: Internet-secure connections to a PAS for OpenEdge instance using HTTPS must have a certificate store for public key certificates on the client (SSL client) and a certificate and key store for private keys and certificates on the PAS for OpenEdge instance (SSL server). For information on configuring both OpenEdge clients and PAS for OpenEdge instances for HTTPS connections, see 'Digital Certificate Management' in *Migrate Classic AppServer Applications to PAS for OpenEdge*. Other SOAP clients and REST clients (such as web browsers) have their own tools for managing certificate stores.

[*username*:*passphrase*@]

Specifies any user login credentials required to connect to the specified ABL web application using the specified *transport* on the PAS for OpenEdge instance, according to the authentication model configured for the *transport*. If the authentication model is Anonymous, no user credentials are required. If the model is HTTP Basic or HTTP Forms, you must provide a valid username (*username*) and password (*passphrase*) known to the user account systems configured for the *transport*. Also, SOAP and REST clients often use an API to pass the user credentials in an authentication header of the HTTP message, in which case you do not need to specify them in the URL. Note that any user credentials you specify in the URL are passed in this authentication header of the HTTP message. However, to protect them, always use HTTPS for the life of the connection.

Note: The APSV and SOAP transports support only Anonymous and HTTP Basic authentication (not HTTP Forms).

For OpenEdge clients using the APSV transport (ABL and Open Clients), you can separately pass the same, or different, user credentials in a client-principal object, or if the connection is session-managed, as connection parameters to the PAS for OpenEdge Connect procedure. In this case, the ABL application on the server typically manages the authentication of these separate credentials at a secondary level, **after** authentication succeeds on the server.

For more information, see the topics about using a client-principal and the Connect procedure in 'Access the connection ID on a session-managed client' in *Migrate Classic AppServer Applications to PAS for OpenEdge*.

host

Specifies the name or domain of the PAS for OpenEdge host.

port

Specifies the port for the host connection. Typically, HTTP uses 8810 and HTTPS uses 8811.

[*web-application/*]

Specifies the OpenEdge ABL web application with which to connect. If *web-app-name* is not specified, a connection to the default OpenEdge ABL web application (ROOT) is used.

transport

Specifies the transport for the connection using the following syntax:

Syntax

```
apsv | soap | rest | web
```

apsv

Specifies the APSV transport used by ABL clients and Java and .NET Open Clients.

soap

Specifies the SOAP transport used by both ABL and external SOAP web service clients.

rest

Specifies the REST transport used by REST adapter clients, which primarily include web browser applications and apps running in mobile devices, such as Rollbase® and mobile apps accessing OpenEdge Data Object Services.

web

Specifies the WEB transport used by new REST clients and migrated WebSpeed clients using the compatibility handler.

service-resource

Supported only for the OpenEdge SOAP and REST transports, this specifies a OpenEdge SOAP or REST web service and its associated resource to access using the following syntax:

Syntax

```
wsdl?targetURI=urn: [ SOAPnamespace: ] SOAPsvc-name | RESTsvc-name/resource-path
```

[*SOAPnamespace*:] *SOAPsvc-name*

Specifies a URI that identifies the namespace for SOAP web services defined in the WSDL file (*SOAPnamespace*) and the name of a service (*SOAPsvc-name*) to access, as defined in the WSDL file. *SOAPsvc-name* can also identify the service by itself if there is only one web service defined in the WSDL file.

RESTsvc-name/resource-path

Specifies a URI path that identifies the name of a REST web service (*RESTsvc-name*) and the REST resource (*resource-path*) to access. The *resource-path* can include query parameters for the resource as well.

For more information about the formation of URIs for accessing OpenEdge SOAP, REST, and WEB services, see Develop ABL services.

In addition to specifying the URL for an ABL web application, the client connection must specify the application model that PAS for OpenEdge and its business application is expected to execute. This is done for each transport and client type when it connects to PAS for OpenEdge. It is the same thing as specifying the session model when you connect to the classic AppServer:

- **APSV transport** — For ABL clients, by setting the `-sessionModel` connection parameter of the `CONNECT( )` method; for Java and .NET Open Clients, by setting the appropriate Open Client proxy property.
- **SOAP transport** — The application model must be specified for the OpenEdge SOAP web service when you define it in ProxyGen and is included in the WSDL file that you generate for client access. SOAP clients need to be written only to support the application model for which the web service is defined and for which the implementing business application on PAS for OpenEdge is written to support.
- **REST transport** — The application model is session-free to execute OpenEdge REST web service requests. All REST clients need to be written to support only session-free access.
- **WEB transport** — The application model is session-free to execute OpenEdge WebSpeed client requests. All REST clients need to be written to support only session-free access.

For more information on migrating client connections from accessing classic AppServer services to access corresponding services of PAS for OpenEdge ABL web applications, see the following topics:

- Migrate AIA URLs to use the APSV transport on page 52
- Migrate WSA URLs to use the SOAP transport on page 53
- Migrate REST URLs on page 54
- Use load balancing in PAS for OpenEdge on page 54
- ABL Application Code Migration on page 35

Migrate AIA URLs to use the APSV transport

PAS for OpenEdge does not support the OpenEdge NameServer or the direct connection modes of the classic AppServer. All connections must use a URL that is similar to what was used for the classic AppServer Internet Adapter (AIA).

For example, an ABL client `-URL` connection parameter to the `CONNECT( )` method might be the following for an AIA connection, which is also shown in a connection parameters string that includes the `-sessionModel` connection parameter:

```
"-URL http://host:port/aia/Aia?AppService=asbroker1 -sessionModel Session-free"
```

The equivalent APSV connection parameter for a PAS for OpenEdge connection might be the following:

```
"-URL http://host:port/apsv -sessionModel Session-free"
```

If the *apsv* URI is `apsv`, or omitted, then it connects to the default `ROOT` ABL web application. If *apsv* is `asbroker1WebAppl/apsv`, then it connects to the ABL web application with the name, `asbroker1WebAppl`, which is deployed in the WAR file, `asbroker1WebAppl.war`.

Note: The APSV connection application model is specified as session-free in the same way as the AIA connection session model.

For more information on ABL client (as well as Open Client) connection to PAS for OpenEdge, see 'Connect to a PAS for OpenEdge instance' in *Application Development with PAS for OpenEdge*.

Migrate WSA URLs to use the SOAP transport

How you connect SOAP web service clients varies significantly with the client platform, but they all provide a way to specify a URL to the WSDL file and the web service they are connecting to.

For example, an ABL client `-WSDL` connection parameter to the `CONNECT( )` method might be the following for a classic OpenEdge Web Services Adapter (WSA) connection:

```
"-WSDL http://host:port/wsa/wsa1/wsdl?targetURI=urn:CustomerSvc"
```

The equivalent SOAP connection parameter for a PAS for OpenEdge connection might be the following:

```
"-WSDL http://host:port/soap/wsdl?targetURI=urn:CustomerSvc"
```

If the *soap* URI is `soap` or omitted, then it connects to the default `ROOT` ABL web application. If *soap* is `asbroker1WebAppl/soap`, then it connects to the ABL web application with the name, `asbroker1WebAppl`, which is deployed in the WAR file, `asbroker1WebAppl.war`.

Note: This example uses the same WAR file name as in the previous example of a classic AIA URL migration (see Migrate AIA URLs to use the APSV transport on page 52), instead of using a file name that reflects the original name of the WSA instance, `wsa1`. This is to illustrate that all three web connection transports can be supported in a single ABL web application on PAS for OpenEdge as long as they can all ultimately access the same server ABL code base.

Note also that just as for a connection to the classic WSA, you specify the exact URL for a connection to an OpenEdge SOAP web service when you deploy the web service. This generates the deployment WSDL file that contains the updated connection URL. It might also be possible to deploy the same WSM file to PAS for OpenEdge that you originally created from generating an existing SOAP web service for the classic WSA using ProxyGen. For more information on OpenEdge SOAP web service deployment to PAS for OpenEdge, see Deploy SOAP services in *Manage Progress Application Server (PAS) for OpenEdge*, and Manage SOAP transports in *Manage Progress Application Server (PAS) for OpenEdge with OpenEdge Management*.

For more information on connecting to a SOAP web service from an ABL client, see the topics on creating ABL clients to consume SOAP web services in Develop an ABL SOAP Client.

Migrate REST URLs

The only difference between the classic OpenEdge AppServer and PAS for OpenEdge regarding the URLs used to connect to and access OpenEdge REST and Data Object Services, is allowing access to the default `ROOT` ABL web application on PAS for OpenEdge.

For example, a URL to access an OpenEdge REST web service for the classic AppServer using the OE Web Server might be the following:

```
http://host:port/CustomerMaint/rest/CustomerSvc/CustConnect
```

After migrating the same ABL code to the `asbroker1WebAppl.war` ABL web application on a PAS for OpenEdge instance, you would use this URL:

```
http://host:port/asbroker1WebAppl/rest/CustomerSvc/CustConnect
```

Note: This example uses the same WAR file name as in the previous example of a classic AIA URL migration (see Migrate AIA URLs to use the APSV transport on page 52), instead of using a file name that might match the original REST web application for the classic AppServer on the OE Web Server (`CustomerMaint.war`). This is to illustrate that all three web connection transports can be supported in a single ABL web application on PAS for OpenEdge as long as they can all access the same server ABL code base.

You could also rename the ABL web application to the file name of the original REST web application for the classic AppServer (`CustomerMaint.war`), and use a URL almost identical to the original (shown in the previous example), or after migrating the ABL code to the default `ROOT` ABL web application, you could use this URL:

```
http://host:port/rest/CustomerSvc/CustConnect
```

Note that OpenEdge ABL does not have a native REST `CONNECT( )` method at this time. However, you can test a REST connection through a browser.

For more information about creating and connecting to both REST web services and Data Object Services, see the topics on these services in Develop ABL Services.

Use load balancing in PAS for OpenEdge

The classic AppServer can optionally rely on a NameServer that enables clients to locate AppServer instances by their application service name. Therefore, you do not have to configure the clients with explicit network addresses and ports to access a particular classic AppServer instance. This same NameServer can achieve server fault tolerance and load balancing for a high client load by having access to multiple classic AppServer instances that provide the same application service to clients.

Because it is a web application server, PAS for OpenEdge does not use the OpenEdge NameServer. It does allow the use of a DNS load balancing service and server-side proxies that allow a client to connect to a server resource without having to know the explicit network address and port, thus achieving a similar level of fault tolerance and load balancing using multiple PAS for OpenEdge instances that support the same ABL web application.

All OpenEdge clients must convert to connecting to a URL that identifies both the ABL web application and connection transport supported by the PAS for OpenEdge instances registered with the DNS server. For PAS for OpenEdge REST and SOAP clients, the URL changes similar to the changes required when you move classic AppServer REST or WSA Adapters to a different Web server.

Note: For more information about load balancing configuration options for PAS for OpenEdge, see Load Balance PAS for OpenEdge Instances.

4

Migrate Server Configuration and Management

Because PAS for OpenEdge runs ABL business applications in a web server environment, PAS for OpenEdge instance configuration and management is very different from the classic OpenEdge AppServer. This content describe some of the most significant differences.

For details, see the following topics:

- Manage server instances
- Migrate classic AppServer properties
- Application packaging and installation
- Performance and resource tuning
- Classic AppServer features not applicable to PAS for OpenEdge
- Security in classic AppServer versus PAS for OpenEdge

Manage server instances

The classic AppServer is a multi-instance architecture. The OpenEdge installation contains the core AppServer libraries, configuration files, and other support files. You create and manage an instance using one of the classic AppServer administration tools, such as OpenEdge Management or the `mergeprop` command-line tool. Each instance has its own configuration and represents one ABL application (identified by a set of configuration files, r-code files, configuration procedures, and classic AppServer brokers and agents). You may not have thought of it this way, but the template classic AppServers (asbroker1, restbroker1, and others) are, in fact, classic AppServer instances supplied as development examples. Actual deployed ABL applications do not typically use those development instances.

PAS for OpenEdge also works with instances but with a difference. The PAS for OpenEdge core server is still located in the OpenEdge installation, under the `$DLC/servers/pasoe` directory. It contains the shared libraries, utilities, and default templates used by instances. Unlike the classic AppServer, the majority of PAS for OpenEdge libraries, configuration files, default applications, and so forth are contained within the instance itself, rather than being scattered throughout the OpenEdge installation directory tree.

A PAS for OpenEdge configuration is divided between the Progress Application Server installation directory, common to all Progress web products, and the OpenEdge product that is deployed into it. The command-line utility for managing PAS for OpenEdge instances is TCMAN, which supports a wide variety of operations that range from creating, configuring, and deleting an instance to controlling the deployment of ABL web applications (and other web applications) contained within an instance. For more information about TCMAN, see TCMAN in *Manage Progress Application Server (PAS) for OpenEdge*. OpenEdge Management also supports some offline operation of a PAS for OpenEdge instance through TCMAN. For more information, see Manage PAS for OpenEdge data in *Manage Progress Application Server (PAS) for OpenEdge with OpenEdge Management*.

Configuration of ABL applications is managed through the following command-line utilities, which are in the `/bin` directory of the instance:

- **`oeprop`**—A command-line utility for secure installations that allows direct manipulation of the `conf/openedge.properties` file in the same general way that the `mergeprops` utility manipulates the classic AppServer's `ubroker.properties` file. For more information, see OEPROP.
- **`deploySOAP`**—A command-line utility for secure installations that allows the incremental deployment of a SOAP description (`.wsm`) file to an ABL application's OpenEdge ABL web application, much like the WSAMAN utility does for the classic Web Services Adapter (WSA). For more information, see Deploy SOAP services.
- **`deployREST`**—A command-line utility for secure installations that allows the incremental deployment of a REST service description (`.paar`) file to an ABL application's OpenEdge ABL web application, much like the RESTMAN utility does for classic OpenEdge REST Web applications. For more information, see Deploy REST services.

Configuration of ABL applications can be optionally managed through the REST APIs of the OpenEdge Manager Web application (`oemanager.war`) in unsecured installations on a PAS for OpenEdge development server.

The OpenEdge Manager uses the REST APIs to perform online administration of ABL applications and their attending web applications, transports, the session manager, and multi-session agents. The OpenEdge Manager REST APIs are open and available for OpenEdge developers to use in their own management tools. For more information, see Use Swagger UI to explore management REST APIs

---

Caution: Do not use remote web application management applications such as the Tomcat `manager.war` or OpenEdge `oemanager.war` in a secure internet web server installation such as a PAS for OpenEdge production server.

Similar to the classic AppServer, you starts with creating a PAS for OpenEdge instance. In a PAS for OpenEdge development server installation, a sample instance is provided for you in the OpenEdge `$WRKDIR` location. This sample instance is defined in the `openedge.properties` configuration file with the ABL application name `oepas1`. Within `oepas1`, is the default ABL web application deployed as `/webapps/ROOT`. You can incrementally deploy SOAP and REST web services, and connect OpenEdge clients to this default ABL web application. You can also create your own ABL web application and add it to the `oepas1` ABL application.

The following set of tasks compare the creation of instances for the classic OpenEdge AppServer and PAS for OpenEdge.

The general set of steps for creating an instance in the classic production AppServer is:

1. Create the instance and give it a name.
2. Deploy `.p` and/or `.r` files.
3. Configure the classic AppServer instance (which implicitly defines an ABL application).
4. Optionally deploy a web server and install Adapters, for example:
   a. Download and install Tomcat.
   b. Configure Tomcat.
   c. Optionally change Tomcat's debug configuration to a secure production type.
      a. Optionally install and configure an AIA adapter web application.
      b. Optionally install and configure a WSA web application, and deploy and configure zero or more SOAP web service descriptors (`.wsm` files).
      c. Optionally install and configure a REST web application (exported from PDSOE), and incrementally deploy zero or more REST web service definitions (`.paar` files).

When the PAS for OpenEdge production server product is installed, no default sample instance is installed for security reasons. Therefore, you follow this sequence of steps:

1. Create the PAS for OpenEdge instance, and give it a name (which implicitly defines an ABL application with the instance name located in the *`instance-name`*`/ablapps` directory).
2. Configure the core Tomcat server for the instance.
3. Deploy `.r` files. (`.p` files cannot be compiled on a PAS for OpenEdge production server.)
4. Deploy the ABL application's OpenEdge ABL web application (one or more).
5. Configure the ABL application and its components, including the ABL session manager and multi-session agents.
6. For each deployed ABL web application:
   a. Configure the APSV transport (enable or disable).
   b. Configure the SOAP transport (enable or disable), and incrementally deploy and configure zero or more SOAP Web service descriptors (`.wsm` files).
   c. Configure the REST transport (enable or disable), and incrementally deploy and configure zero or more REST Web service definitions (`.paar` files).
   d. Configure the WEB transport (enable or disable) and any associated web handlers.

Migrate classic AppServer properties

The classic AppServer keeps the properties for all defined instances in a single properties file located at `$DLC/properties/ubroker.properties`. With those properties, you define a set of property values that serve as the application's default configuration. An administrator installing an application typically defines a new classic AppServer instance and can change those property values to suit a site's specific requirements.

PAS for OpenEdge operates in the same general way, but is different with regards to the following:

- Each PAS for OpenEdge instance embeds its `openedge.properties` file inside the *`instance-name`*`/conf` directory.
- A PAS for OpenEdge instance supports named ABL applications that correspond to the installation directory architecture.
- The properties for an ABL application in PAS for OpenEdge configure its web applications, transports, an ABL session manager, and ABL multi-session agents.

The following is a brief comparison of classic AppServer properties to PAS for OpenEdge properties:

- Many of the properties used by a classic AppServer can be found in PAS for OpenEdge.
- Some classic AppServer properties have no meaning and are not found in PAS for OpenEdge.
- Some classic AppServer properties relate to functionality that is provided by the PAS for OpenEdge installation directory and are managed using core, server-level properties.
- PAS for OpenEdge has new properties do not apply to the classic AppServer.

The following is an overview of how the properties found in the `ubroker.properties` file for a classic AppServer map to properties found in a PAS for OpenEdge instances `openedge.properties` file. The detailed information about individual properties can be found in the respective README files (`ubroker.properties.README` and `openedge.properties.README`).

There are certain rules for reading property names:

- All classic AppServer property names do not supply a group (a `[...]` name as the properties can exist in any `[Ubroker…]` group).
- All PAS for OpenEdge property names use a name-space declaration that tracks the group hierarchy found in the `openedge.properties` file. For example:
  - `AppServer.Agent [AppServer.Agent]`
  - `AppServer.Agent [AppServer.Agent]`
  - `AppServer.Agent.oepas1 [AppServer.Agent.oepas1]`
  - `AppServer.Agent.wrkdir [AppServer.Agent] wrkdir`
  - `AppServer.Agent.oepas1.wrkdir [AppServer.Agent.oepas1] wrkdir`
- PAS for OpenEdge groups include:
  - **`[AppServer]`** — Properties that apply across the server instance
  - **`[AppServer.Agent]`** — Properties that apply across all multi-session agents in the server instance (inherits from nothing)

- **[AppServer.Agent.*appname*]** — Multi-session agent properties that apply only to the ABL application named *appname* (inherits from [AppServer.Agent])
- **[AppServer.SessMgr]** — Properties that apply across all ABL application session managers in the server instance (inherits from nothing)
- **[AppServer.SessMgr.*appname*]** — ABL application session manager properties that apply only to the ABL application named *appname* (inherits from [AppServer.SessMgr])
- **[*appname*]** — Properties that apply only to the ABL application's web applications (inherits from nothing)
- **[*appname*.*web-app-name*]** — Properties that apply only to the ABL application's web application named *web-app-name* (inherits from [*appname*])
- **[*appname*.*web-app-name*.APSV]** — Properties that apply only to the ABL application's web application named *web-app-name* and to its APSV transport (inherits from [*appname*.*web-app-name*])
- **[*appname*.*web-app-name*.SOAP]** — Properties that apply only to the ABL application's web application named *web-app-name* and to its SOAP transport
- **[*appname*.*web-app-name*.REST]** — Properties that apply only to the ABL application's web application named *web-app-name* and to its REST transport
- **[*appname*.*web-app-name*.WEB]** — Properties that apply only to the ABL application's web application named *web-app-name* and to its WEB transport

The following table list the classic AppServer properties and how they correspond to a PAS for OpenEdge instance properties:

Table 10: Mapping AppServer to PAS for OpenEdge instance properties

Classic AppServer property	PAS for OpenEdge property
4glSrcCompile	N/A
allowRuntimeUpdates	AppServer.allowRuntimeUpdates
appServerKeepaliveCapabilities	N/A
appserviceNameList	N/A
autoStart	N/A
autoTrimTimeout	N/A
brkrDebuggerEnabled	N/A
brkrDebuggerKeyAlias	N/A
brkrDebuggerKeyAliasPassword	N/A
brkrDebuggerPassphrase	N/A
brkrDebuggerPortNumber	N/A
brkrDebuggerSSLEnable	N/A

Classic AppServer property	PAS for OpenEdge property
brkrDebuggerUseBrokerAlias	N/A
brkrLogAppend	N/A
brkrLogEntries	N/A
brkrLogEntryTypes	N/A
brkrLogThreshold	N/A
brkrLoggingLevel	N/A
brkrNumLogFiles	N/A
brokerLogFile	N/A
certStorePath	N/A
classMain	N/A
collectStatsData	AppServer.Agent.collectStatsData
compressionEnable	N/A
connectingTimeout	See catalina.properties
controllingNameServer	N/A
debuggerEnabled	N/A
defaultService	N/A
description	N/A
environment	N/A
flushStatsData	AppServer.Agent.flushStatsData
groupName	N/A
hostName	N/A
infoVersion	AppServer.Agent.infoVersion
initialSrvrInstance	AppServer.SessMgr.numInitialAgents
ipver	AppServer.SessMgr.ipver
keyAlias	AppServer.Agent.keyAlias
keyAliasPasswd	AppServer.Agent.keyAliasPasswd

Classic AppServer property	PAS for OpenEdge property
keyStorePasswd	AppServer.Agent.keyStorePasswd
keyStorePath	AppServer.Agent.keyStorePath
maxClientInstance	See openedge.properties.README
maxSrvrInstance	AppServer.SessMgr.maxAgents
minSrvrInstance	N/A
noSessionCache	AppServer.Agent.noSessionCache
operatingMode	N/A
password	N/A
portNumber	See catalina.properties
priorityWeight	N/A
PROPATH	AppServer.Agent.PROPATH
registerNameServer	N/A
registrationMode	N/A
registrationRetry	N/A
requestTimeout	AppServer.SessMgr.requestWaitTimeout
serverASKActivityTimeout	N/A
serverASKResponseTimeout	N/A
serverLifespan	N/A
serverLifespanPadding	N/A
sessionTimeout	AppServer.Agent.sessionTimeout
srvrActivateProc	AppServer.Agent.sessionActivateProc
srvrConnectProc	AppServer.Agent.sessionConnectProc
srvrDeactivateProc	AppServer.Agent.sessionDeactivateProc
srvrDisconnProc	AppServer.Agent.sessionDisconnProc
srvrExecFile	AppServer.SessMgr.agentExecFile
srvrExecutionTimeLimit	N/A

Classic AppServer property	PAS for OpenEdge property
srvrLogAppend	N/A
srvrLogEntryTypes	AppServer.SessMgr.agentLogEntryTypes
srvrLogFile	AppServer.Agent.agentLogFile
srvrLogThreshold	AppServer.SessMgr.agentLogThreshold
srvrLogWatchdogInterval	N/A
srvrLoggingLevel	AppServer.SessMgr.agentLoggingLevel
srvrMaxPort	AppServer.Agent.agentMaxPort
srvrMinPort	AppServer.Agent.agentMinPort
srvrNumLogFiles	AppServer.SessMgr.agentNumLogFiles
srvrSelectionScheme	N/A
srvrShutdownProc	AppServer.Agent.sessionShutdownProc
srvrStartupParam	AppServer.Agent.sessionStartupParam
srvrStartupProc	AppServer.Agent.sessionStartupProc
srvrStartupProcParam	AppServer.Agent.sessionStartupProcParam
srvrStartupTimeout	N/A
sslAlgorithms	AppServer.Agent.sslAlgorithms
sslEnable	AppServer.Agent.sslEnable
userName	N/A
uuid	AppServer.Agent.uuid
workDir	N/A
N/A	AppServer.statusEnabled
N/A	AppServer.collectMetrics
N/A	AppServer.applications
N/A	AppServer.Agent.agentShutdownProc
N/A	AppServer.Agent.agentStartupProc
N/A	AppServer.Agent.agentStartupProcParam

Classic AppServer property	PAS for OpenEdge property
N/A	*appname*.webApps
N/A	*appname*.SessionMgr
N/A	*appname*.*web-app-name*.adapterEnabled
N/A	*appname*.*web-app-name*.SOAP.adminEnabled
N/A	*appname*.*web-app-name*.SOAP.adminSoapAction
N/A	*appname*.*web-app-name*.SOAP.debugClients
N/A	*appname*.*web-app-name*.SOAP.wsdlEnabled
N/A	*appname*.*web-app-name*.SOAP.wsaUrl
N/A	AppServer.SessMgr.publishDir

Migrate classic AppServer properties to PAS for OpenEdge using PASPROPCONV

To begin migrating a classic AppServer application to PAS for OpenEdge, use the PASPROPCONV utility to convert the `ubroker.properties` file that supports your classic AppServer application to the `openedge.properties` format that is used by PAS for OpenEdge.

The following topic describes the process of using PASPROPCONV to convert the classic AppServer properties for a single ABL application to the PAS for OpenEdge format: Prepare to move from Classic AppServer to PAS for OpenEdge on page 16.

For the more advanced use case of migrating multiple classic AppServer ABL application properties to PAS for OpenEdge, see the following topic: Migrate multiple classic AppServer properties to PAS for OpenEdge using PASPROPCONV on page 65.

For more information about the PASPROPCONV utility, see "PASPROPCONV" in *Manage Progress Application Server (PAS) for OpenEdge*.

Migrate multiple classic AppServer properties to PAS for OpenEdge using PASPROPCONV

To convert multiple classic AppServer application properties to PAS for OpenEdge, use the PASPROPCONV utility to specify how you want to configure your applications on PAS for OpenEdge. The migration process involves creating new ABL applications and ABL web applications on a PAS for OpenEdge instance to suit your organization's business logic requirements. After you run the PASPROPCONV utility on your classic AppServer applications, merge the exported properties into the PAS for OpenEdge instance.

This topic describes how you migrate multiple classic AppServer ABL applications to PAS for OpenEdge using three example applications, which may contain some analogs to applications in your organization.

For more information about the migration process, see Quick Start: Move Classic AppServer Applications to PAS for OpenEdge on page 13.

To migrate multiple classic AppServer applications to PAS for OpenEdge using PASPROPCONV:

1. Create a PAS for OpenEdge instance (for example, `myProdInst`):

```
proenv>pasman create -Z prod myProdInst
```

   This instance comes with a default ABL application called `myProdInst`, and a ROOT web application that accesses the `myProdInst` ABL application.

   For more information about ABL applications, the ROOT web application, and web applications on PAS for OpenEdge, see "What is PAS for OpenEdge" in *Learn about Progress Application Server for OpenEdge*.

   Note: If you do not require a ROOT web application for your server URL (for example, if you want all web applications to be accessed using named server URLs, rather than through the root (/) URL), then create the instance using the `-Z pas` security model. If the instance already exists and you do not require a ROOT web application, then deploy the `noaccess.war` application in place of ROOT. For more information, see "About security models" in *Manage Progress Application Server for OpenEdge*.

2. Undeploy the default ROOT web application, to remove its association with the default ABL application, `myProdInst`:

```
proenv>myProdInst/bin/tcman undeploy ROOT
```

   This topic assumes that you want to access a migrated ABL application, `inventoryApp`, through the root (/) URL, so you must remove the ROOT application's association with the default ABL application, `myProdInst`, by undeploying ROOT.

3. Create the ABL applications that correspond to your existing clasic AppServer applications on the PAS for OpenEdge instance using `TCMAN deploy`. Use `TCMAN deploy` while specifying the ABL application and ABL web application name to create the applications on the PAS for OpenEdge instance, for example:

```
proenv>myProdInst/bin/tcman deploy -a ROOT $DLC/servers/pasoe/extras/oeabl.war
inventoryApp
proenv>myProdInst/bin/tcman deploy -a finance $DLC/servers/pasoe/extras/oeabl.war
financeApp
proenv>myProdInst/bin/tcman deploy -a shipping $DLC/servers/pasoe/extras/oeabl.war
shippingApp
```

   In this example, the PAS for OpenEdge ABL applications are called `inventoryApp`, `financeApp`, and `shippingApp`. The `inventoryApp` ABL application is associated with the default ROOT web application, while the `financeApp` and `shippingApp` applications are associated with ABL web applications with related names (`finance` and `shipping`, respectively).

   ABL applications are accessed through the ABL web application using a server URL, for example:

   - The `financeApp` ABL application is accessed through the `finance` web application at `http://`*`hostname`*`:`*`port`*`/finance/`
   - The `shippingApp` ABL application is accessed through the `shipping` web application at `http://`*`hostname`*`:`*`port`*`/shipping/`
   - The `inventoryApp` ABL application is accessed through the default ROOT web application at `http://`*`hostname`*`:`*`port`*`/`

4. Export the `ubroker.properties` for each classic AppServer ABL application using PASPROPCONV.

a. If you are migrating properties of a classic AppServer application to the ROOT ABL web application on the PAS for OpenEdge instance (as in the example `inventoryApp` created in Step 2), specify the `ubroker.properties` file location, ubroker name, and ABL application name in the PASPROPCONV command:

```
proenv>paspropconv --ubrokerPropsFile path/to/inventory/ubroker.properties
--ubrokerName UBroker.AS.inventory_prod  --pasoeAppName inventoryApp
```

The command exports the `ubroker.properties` for use in the ROOT web application that is deployed to the `inventory` ABL application, but is accessed through the ROOT web application server URL (/).

b. For each of the ABL applications on the PAS for OpenEdge instance that you want to run in an ABL web application that is specified using the server URL, run the PASPROPCONV command specifying the ABL application name and the ABL web application name that you specified at deployment, for example:

```
proenv>paspropconv --ubrokerPropsFile path/to/finance/ubroker.properties
--ubrokerName UBroker.AS.finance_prod  --pasoeAppName financeApp --pasoeWebAppName
 finance
proenv>paspropconv --ubrokerPropsFile path/to/shipping/ubroker.properties
--ubrokerName UBroker.AS.shipping_prod  --pasoeAppName shippingApp --pasoeWebAppName
 shipping
```

The command exports the `ubroker.properties` file for for each classic AppServer application for use in the named PAS for OpenEdge web application URL (`/finance` and `/shipping`, respectively).

Running PASPROPCONV on your classic AppServer applications produces a several files, including:

- `inventoryApp.inventory_prod.oemerge`
- `financeApp.finance_prod.oemerge`
- `shippingApp.shipping_prod.oemerge`

A corresponding Set Environment script for each application is also produced.

5. Move each Set Environment script to the `myProdInst/bin` directory.

6. Review and modify the `oemerge` files according to your organization's requirements. For more information, see "Review and update the oemerge file" in Prepare to move from Classic AppServer to PAS for OpenEdge.

7. From the `myProdInst/bin` directory, run the OEPROP utility on each `oemerge` file, for example:

```
proenv>oeprop -f path/to/inventoryApp.inventory_prod.oemerge
proenv>oeprop -f path/to/financeApp.finance_prod.oemerge
proenv>oeprop -f path/to/shippingApp.shipping_prod.oemerge
```

8. Review the `myProdInst/conf/openedge.properties` file, and verify that the classic AppServer properties were merged for each application. To verify this, ensure that each ABL application has a section in `openedge.properties` that contains its associated ABL web application (in the `webApps` property), shown in the following entries:

```
[inventoryApp]
 webApps=ROOT

[financeApp]
  webApps=finance
```

```
[shippingApp]
 webApps=shipping
```

Application packaging and installation

PAS for OpenEdge supports a variety of packaging and installation options, from installing applications into an existing PAS for OpenEdge instance to installing entire PAS for OpenEdge instances based on an existing installation. This flexibility is because an ABL application is always installed as part of an OpenEdge ABL web application deployed to the PAS for OpenEdge web server. At the same time, this web application can support all SOAP and REST web service clients, as well as ABL and Open Client access, to the same ABL application. Finally, the ABL application, itself, can be supported by multiple OpenEdge ABL web applications on a single PAS for OpenEdge instance.

The following topics describe in greater detail how PAS for OpenEdge packaging and installation differs from that of the classic AppServer.

Distribution packaging

The classic AppServer does not have a defined directory structure or location where ABL application files should be deployed, or where its working and temporary files should be located. With this lack of structure, it is up to the ABL application developer to determine how to manage scalability and extensibility.

Because it is based on the Apache Tomcat server platform, PAS for OpenEdge follows the conventions common to the Tomcat environment for deploying ABL applications as web applications. Even though an ABL application cannot be considered a fully scalable or extensible application when comparing it to a standard Tomcat web application, it can be designed, packaged, and distributed so that it is scalable and extensible with a minimal amount of work. PAS for OpenEdge, through the use of the PAS for OpenEdge installation server functionality, offers additional deployment packaging and deployment options to ABL application developers that does not exist for standard Tomcat and web application developers.

The classic AppServer supports REST and SOAP services through external Adapters. This same support exists for PAS for OpenEdge, however services are deployed to different locations and there are different utilities for managing the deployment. PAS for OpenEdge incorporates the functionality of the classic AppServer's external adapters into a single deployable web application, which can be the same web application in which the ABL application is deployed. The tools to deploy REST or SOAP services continue to include OpenEdge Management and OpenEdge Explorer, but new command-line tools (`deploySOAP` and `deployREST`) are used to correspond with the architectural changes in PAS for OpenEdge.

In addition, PAS for OpenEdge supports:

- Deploying individual SOAP service descriptors (`.wsm` files) into an existing OpenEdge ABL web application.
- Deploying individual REST service definitions (`.paar` files) into an existing OpenEdge ABL web application.
- Deploying an entire OpenEdge ABL web application, including pre-deployed SOAP and REST web services, into an existing PAS for OpenEdge instance.
- Deploying an entire PAS for OpenEdge instance, including pre-deployed OpenEdge ABL web applications, into an existing OpenEdge installation.

Note: Unlike the classic AppServer, PAS for OpenEdge instances are not deleted when OpenEdge is uninstalled. Therefore, the instances and their complete deployment and configuration can simply be added to a fresh OpenEdge installation. For more information, see the TCMAN register command in the *PAS for OpenEdge Administration Tools Reference*.

Incremental SOAP services

A SOAP web service is deployed using a SOAP descriptor file (`.wsm`) that calls ABL service interface APIs in the ABL application. To deploy incremental SOAP services into an existing PAS for OpenEdge instance's ABL web application, package the `.wsm` file and any ABL `.p` or `.r` language files that support the service interface's API. After the package is available on the PAS for OpenEdge instance's host server you will:

1. Unpack the files into a temporary location.
2. Use the PAS for OpenEdge instance's `deploySOAP` command-line utility to deploy the SOAP descriptor file into an existing OpenEdge ABL web application. For more information, see Deploy SOAP services.
3. Move any ABL `.p` or `.r` files into one of the ABL application's `PROPATH` locations.

Note: The `PROPATH` location can include the OpenEdge ABL web application's `WEB-INF/openedge` directory tree.

Incremental REST services

A REST web service is deployed using a REST service definition file (`.paar`) that calls ABL service interface APIs in the ABL application. To deploy incremental REST service definitions into an existing PAS for OpenEdge instance's ABL web application, package the `.paar` file and any ABL `.p` or `.r` language files that support the service interface's API. Once the package is available on the PAS for OpenEdge instance's host server you will:

1. Unpack the files into a temporary location.
2. Use the PAS for OpenEdge instance's `deployREST` command-line utility to deploy the REST service definition file into an existing ABL web application. For more information, see Deploy REST services.
3. Move any ABL `.p` or `.r` files into one of the ABL application's `PROPATH` locations.

Note: The `PROPATH` location can include the OpenEdge ABL web application's `WEB-INF/openedge` directory tree.

Basic web application packaging

You can deploy an ABL application as one or more OpenEdge ABL web applications. One OpenEdge ABL web application is packaged as one Web Application Archive (`.war`) file that follows the layout outlined previously (see ABL application structure on page 36). For improved performance and resource consumption, deploy an ABL application using a single OpenEdge ABL web application—making the full application accessible from a single server URL. There are situations where deploying an ABL application as multiple OpenEdge ABL web applications (where the application is accessible across multiple server URLs) is the correct path to follow. Because using multiple OpenEdge ABL web applications slows performance and consumes more OS process resources, use this approach only when:

- Your OpenEdge ABL web application is designed as a set of discrete, individually deployed, services, where the OpenEdge ABL web application encapsulates "service set" within your ABL application.
- You must avoid interrupting classic AppServer clients until they can be upgraded to use a single URL.

When deploying OpenEdge ABL web applications, you can pre-deploy and configure SOAP, REST, and Spring Security. You also have the option to deploy the SOAP, REST, WEB, and APSV service interface API's `.p` and `.r` files as part of each web application's `WEB-INF/openedge` directories. Using the optional embedding of `.r` and `.p` modules in the web application, you ensure that the version of the SOAP, REST, WEB, and APSV service interfaces remains consistent with their transport (URI) definitions. You may need to change the ABL application's `PROPATH` to include the OpenEdge ABL web application's `WEB-INF/openedge` path.

PAS for OpenEdge instance packaging

PAS for OpenEdge can be packaged and deployed as a complete, pre-configured instance. The instance can be unpacked in an OS file system and reconfigured for the local OpenEdge installation and for the site's requirements. Briefly stated, you do the following:

1. Create a PAS for OpenEdge instance that represents the deployment package.
2. Deploy and configure the OpenEdge ABL web applications that will make up the ABL application.
3. Deploy and configure any SOAP and REST web services into the OpenEdge ABL web applications.
4. Deploy the ABL application's `.p` and/or `.r` files as part of the *`instance-name`*`/openedge` directory, or an OpenEdge ABL web application's `openedge` directory (for example, *`instance-name`*`/webapps/ROOT/WEB-INF/openedge`) .
5. Install any application tailoring scripts into the *`instance-name`*`/bin` directory.

   Note: On UNIX, script files must be set to executable because archive files do not preserve file permissions.

6. Supply any OS process environment files containing the environment variables needed by your application.
7. Configure the PAS for OpenEdge server default features and properties. See TCMAN feature and TCMAN config for more information.

To do the packaging, use the Java `jar` utility or the WinZip (or equivalent) utility to create a single archive file that starts with the instance's root (`CATALINA_BASE`) directory.

Note: If you must, you can deploy the ABL application's `.p` and `.r` files anywhere on the OS's file system, just like in a classic AppServer. However, you lose most, if not all, of the benefits of the PAS for OpenEdge infrastructure for deployment, updates, and security.

Installation process and updates

The installation of an ABL application in PAS for OpenEdge depends on the packaging described in the previous topics. These topics focus on the PAS for OpenEdge installation.

Note: All installations depend on having installed **either** the Progress Dev AS for OpenEdge (*development server*) **or** the Progress Production AS for OpenEdge (*production server*) products.

Install a prepackaged PAS for OpenEdge instance

This type of installation deploys a packaged PAS for OpenEdge instance created according to the procedure described in PAS for OpenEdge instance packaging on page 70.

1. Load the `.jar` or `.zip` file that contains the packaged PAS for OpenEdge instance.
2. Set the current working directory to the location where the named instance will be located.
3. Use the `expand` utility found in `$DLC/servers/pasoe/bin/expand.{sh | bat}` to unpack the instance.
4. Use the `$DLC/servers/bin/tcman.{sh | bat} register` command to link the new instance with the OpenEdge installation.
5. If this is a Windows system, and the instance will be run as a Windows service, use the `tcman.bat service create` command to make a new Windows service.
6. If this is a UNIX system, set the correct security level (`{ production | standard | development }`) using the `$DLC/servers/pasoe/bin/openedge_tailor` command, specifying **only** the `security-mode` option.
7. Use the `tcman.{sh | bat} config` command to assign new network (`http/https/ajp13`) ports.
8. Use the `tcman.{sh | bat} feature` command to enable or disable server features according to the site requirements.
9. Use the `oeprop` utility to configure the ABL application's properties according to site requirements.
10. Use a text editor and configure each ABL web application's security according to the site requirements. For more information, see Secure PAS for OpenEdge instances in *Manage Progress Application Server (PAS) for OpenEdge*.

Install a prepackaged ABL web application

This type of installation extends an existing PAS for OpenEdge instance. While deployment of general web applications can be performed online or offline, the implementation of the ABL web applications indicates that the installation is best performed offline.

To install a prepackaged ABL web application:

1. Load the `.war` file containing the prepackaged ABL web application.
2. Run the instance's `tcman.{sh | bat} deploy` command to load the web application into the *`instance-name`*`/webapps` directory. For more information, see Deploy an ABL web application.
3. Run the `oeprop` utility to configure the web application in the ABL application's `openedge.properties` file.
4. Run the `oeprop` utility to configure the web application's APSV, SOAP, REST, and WEB transports according to site requirements.
5. Use a text editor and configure the ABL web application's security according to the site requirements. For more information, see the Secure PAS for OpenEdge instances topics in *Manage Progress Application Server (PAS) for OpenEdge*.
6. Deploy any `.p` or `.r` code required to support the new ABL web application.

a. If the ABL `.p` or `.r` code is packaged inside the ABL web application, use the `oeprop` utility to add its path to the ABL application's `PROPATH`.

b. If the ABL `.p` or `.r` code is packaged inside the ABL web application but needs to be moved elsewhere, move it and update the ABL application's `PROPATH` with its new location.

c. If the ABL `.p` or `.r` code is packaged externally, expand that distribution to its new location and update the ABL application's OpenEdge ABL web application.

Note: If you must, you can deploy the ABL application's `.p` and/or `.r` files anywhere on the OS's file system, just like in a classic AppServer. However, you lose most, if not all, of the benefits of the PAS for OpenEdge infrastructure for deployment, updates, and security.

Update an existing installation, ABL web application, or service

When you need to update PAS for OpenEdge, there are different processes to follow:

- When you update to a later release of OpenEdge, all PAS for OpenEdge instances automatically picks up the new shared libraries and utilities found in the PAS for OpenEdge installation directory in `$DLC/servers/pasoe`.
- It is rare that an ABL web application requires an update because it does not contain patchable code. If patches to the security configuration files or other static files exist, then Progress recommends that you develop an automated script that can make the appropriate file substitutions and additions.
- To update a REST or SOAP web service, use the same process you used with the classic WSA for SOAP services and the classic REST Manager for REST services: undeploy the old service, deploy the new service, and configure the new service.

Install remote administration

The PAS for OpenEdge product runs in a secured configuration unless otherwise configured. Therefore, it does not automatically include the remote web applications used by OpenEdge Management (for Tomcat and OpenEdge components). All Tomcat and OpenEdge administration can be performed locally using a JMX console—the administrative web applications offer no additional functionality. If the site's policies allow remote administration, then you can add these administrative web applications to a PAS for OpenEdge instance at any time by doing the following:

1. Choose which remote administration web application to deploy: Tomcat's `manager` or OpenEdge's `oemanager`.
2. Deploy the remote administration web application from `$DLC/servers/pasoe/extras` using the TCMAN utility's `deploy` command. For more information, see Deploy an ABL web application.
3. Configure the remote administration's security according to the site's policies, using a text editor to alter the `web.xml` and Spring Security configuration files. For more information, see Secure PAS for OpenEdge instances in *Manage Progress Application Server (PAS) for OpenEdge*.

Performance and resource tuning

Performance and resource tuning a PAS for OpenEdge instance is not similar to tuning a classic AppServer. It is more closely aligned with tuning a web server, but not exactly, because of ABL language support. For information about how to begin tuning a PAS for OpenEdge instance to best support your ABL application, see Tune PAS for OpenEdge instances.

Classic AppServer features not applicable to PAS for OpenEdge

In general, the tools and processes you have for a classic AppServer do not apply to PAS for OpenEdge. The entire architecture is different, thus requiring new and different tools and methods. Some of the features associated with the classic AppServer and Adapter products that are not applicable to PAS for OpenEdge include:

- ASBMAN
- NSMAN
- WSAMAN
- RESTMAN
- `ubroker.properties`
- NameServer
- WSA
- REST Remote Administration Agent (OERM)
- AdminServer
- Classic AppServer log files

The following topics describe other aspects of working with the classic AppServer that do not apply to PAS for OpenEdge.

Development process tooling

An ABL application contains the same basic components in both the classic AppServer and PAS for OpenEdge, but it should be packaged and distributed in a way that best suits the architecture of PAS for OpenEdge. Therefore, existing scripts, build tools, and so forth, that package a classic AppServer's ABL application do not work with PAS for OpenEdge. You can continue packaging and distributing your ABL application as if it were for a classic AppServer, but you will lose some of PAS for OpenEdge's value.

Note: If you must, you can deploy the ABL application's `.p` and `.r` files anywhere on the OS's file system, just like in a classic AppServer. However, you lose most, if not all, of the benefits of the PAS for OpenEdge infrastructure for deployment, updates, and security.

Installation packaging and deployment

Most ABL applications have some form of automated script or installation program that makes them easier to install. Depending on the packaging and deployment process chosen for PAS for OpenEdge ABL applications, it may be necessary to change or rewrite these tools. There are a significant number of changes to the location of files and to where configuration properties are managed. Almost all of PAS for OpenEdge is deployed with extensible scripts so that much of the process can be automated.

Site monitoring and administration tools

PAS for OpenEdge is **not** the classic AppServer. Its architecture is different, its Tomcat server subsystems are different, and its execution of the ABL sessions is different. Therefore, all of the classic AppServer's administration and monitoring support is not compatible with PAS for OpenEdge. As a result, any third-party automation tools written to monitor a classic AppServer will no longer be useful, and must be rewritten.

PAS for OpenEdge supports a vastly expanded set of metric data collection that provides a broad picture of server and application health. PAS for OpenEdge supports a number of administrative state queries and operations targeted at discovering what is happening at that moment, with the option to correct certain conditions (hangs, runaways, etc.) without having to restart the entire server.

PAS for OpenEdge provides several administrative and monitoring features:

- **JMX** — The Java standard JMX support for PAS for OpenEdge supplements that of the PAS (Apache Tomcat) JMX administration. JXM gives the administrator full PAS for OpenEdge administration support without exposing the server to dangerous remote administration through web applications. PAS for OpenEdge supports the following JMX tools:
  - JConsole, a GUI tool to explore and manage Java beans, can be accessed on the same machine using the process ID (PID) or by opening ports for remote access. However, opening remote ports can be considered a security risk and is only advised for development. For more information, see Run JConsole remotely and JConsole and JMX.
  - The PAS for OpenEdge `OEJMX` scripts gives the administrator full PAS for OpenEdge administration and monitoring support without exposing the server to dangerous remote administration through web applications. For more information, see Use OEJMX to manage and monitor an instance.
- **REST** — PAS for OpenEdge provides an **optional** REST management API through a web application for OpenEdge Management/OpenEdge Explorer and third-party administration tools. This web application can be secured, but it is not a standard part of the PAS for OpenEdge production server product. For more information, see Use Swagger UI to explore management APIs.
- **Manager** — PAS for OpenEdge provides an **optional** Apache Tomcat Manager Web application for remote administration of the Tomcat server and PAS for OpenEdge applications. See the Apache Tomcat Manager documentation for detailed information. The TCMAN utility provides full text client access to server information. This web application can be secured, but it is not a standard part of the PAS for OpenEdge production server product.
- **HealthScanner** — PAS for OpenEdge provides an **optional** OpenEdge HealthScanner web application that monitors your environment. Enabling data collection for the OpenEdge HealthScanner allows PAS for OpenEdge to collect system data in order to calculate a health score. HealthScanner allows the health of your server to be verified by HTTP request. Additionally, separate HTTP requests can gather summary and detailed information about the data collected. The `OEHealth` script can be used to gather the summary and detailed information without having to enable the web application which can be considered a security risk. For more information, see Use the OpenEdge HealthScanner.

- **Shutdown port** — A shutdown port is an unused port number used for the shutdown of Tomcat. If you are creating a PAS for OpenEdge instance on a Windows machine, then this is a mandatory field. UNIX platforms may disable a shutdown port and use signals to halt a running server. Windows platforms, which lack an effective signaling mechanism, require a valid port number.

Security in classic AppServer versus PAS for OpenEdge

By industry definition, the classic AppServer does not meet security requirements and best practices. For example, you would never deploy a classic AppServer as an Internet server. Whatever security an OpenEdge application has is from the diligent efforts of the OpenEdge developers who have written the tools and installation processes around it.

PAS for OpenEdge provides two products:

- An unsecured Development server product
- A secured Production server product

The two PAS for OpenEdge products are almost identical, with the differences being the security of the configuration. The goal for PAS for OpenEdge in the production server product is to meet 95% of the recommended security best practices for an Apache Tomcat server. The remaining 5% is something either the production administrator is required to do according to the company's policies, or the developer does based on the constraints imposed by their application.

The following is a summary of the Production server product's security configuration:

- Removal of the ABL compiler, preventing any unauthorized source code access
- Removal of all remote administration web applications that can be targeted by intruders
- Core server configuration with removal of unsecured debug features, such as auto-deployment
- UNIX directory and file permission settings
- Additional security *valves* (in other words, server request filters)
- Full administrative capabilities through secure local utilities, such as command-line tools and JMX access

For more information, see Development vs production instances.

Spring Security in PAS for OpenEdge

For the most part, the classic AIA and WSA do not have security built into them, and require manual configuration of the web application's descriptor file (`web.xml`) in order to use the servlet container's security features. In PAS for OpenEdge, the security for the APSV (formerly AIA) and SOAP services was taken over by the Spring Security framework. All security that has previously been implemented for the AIA and WSA can be migrated to the Spring Security's container security.

Classic REST and Data Object web applications already include Spring Security as their primary security framework. This continues in PAS for OpenEdge's ABL web applications. The template files for Spring Security have changed, which requires you to manually port the classic REST and Data Object web application configuration into the configuration files of ABL web applications.

For more information on ABL web application support for Spring Security, see the topics on security in *Manage Progress Application Server (PAS) for OpenEdge* such as Enable ABL application authentication.

5

Migrate Classic WebSpeed Applications

PAS for OpenEdge is available as a host for WebSpeed applications. You can migrate existing WebSpeed applications to PAS for OpenEdge, or you can use the Progress Developer Studio for OpenEdge to develop new WebSpeed applications that run on PAS for OpenEdge.

This topic describes migrating an existing (classic) WebSpeed applications. For information about developing new web applications for PAS for OpenEdge using the WEB transport, see Application servers, web applications, and services in *Develop ABL services*.

This task-based video reviews the most common steps needed to migrate WebSpeed applications to PAS for OpenEdge.

To migrate a WebSpeed application to a PAS for OpenEdge instance, you must do the following:

1. Move the application's static files to a specific folder in the instance.
2. Update the PROPATH for the instance to include the folders that contain the application's r-code.
3. Enable the WEB transport on the instance and use the compatibility handler in the `openedge.properties` file.

Static files

A PAS for OpenEdge instance requires the static files that support a WebSpeed application to be in a particular location in the instance's directory structure. Static files include images and HTML files.

The location of the static files for the default web application is:

```
instance_name/webapps/ROOT/static
```

If you deploy another application, then the default location for its static files is:

```
instance_name/webapps/webapp_name/static
```

where *webapp_name* is the name of the WebSpeed application or ABL web application.

r-code

To enable a PAS for OpenEdge instance to find a WebSpeed application's r-code, add any folders that contain WebSpeed application r-code files to the instance agent's PROPATH.

The default location for r-code is:

```
instance_name/openedge
```

PROPATH is set in the *instance_name*/conf/openedge.properties file. For example:

```
[AppServer.Agent]
PROPATH=${CATALINA_BASE}/openedge,${DLC}/tty,${DLC}/tty/netlib/OpenEdge.Net.pl
```

Note: CATALINA_BASE is an environment variable that resolves to *instance_path*.

Update openedge.properties to enable WEB transport

Ensure that the *instance-name*/conf/openedge.properties has the WEB transport enabled and the defaultHandler is set to OpenEdge.Web.CompatibilityHandler. For example:

```
[abl_app_name.ROOT.WEB]
 adapterEnabled=1
 defaultCookieDomain=
 defaultCookiePath=
 defaultHandler=OpenEdge.Web.CompatibilityHandler
```

Migration Notes

Before you migrate an existing WebSpeed application to an instance of PAS for OpenEdge, note the following:

- A PAS for OpenEdge instance must be created from an OpenEdge 11.6 or later release because that is when the `OpenEdge.Web.CompatibilityHandler` was introduced. For information about creating, starting, and stopping instances, see Create and Configure PAS for OpenEdge instances.
- R-code should be located on the same machine where the PAS for OpenEdge instance is running. R-code on network-mapped drives can cause issues with performance and permissions.
- Recompile r-code only if it was generated in OpenEdge 10.*x* or earlier releases.
- WebSpeed applications that use HTML mapping are not supported.
- WebSpeed applications with a modified `web-disp.p` and supporting files will **not** run without making changes to the default WebHandler.
- WSASP, WSISA, NSAPI, and CGIIP messengers are not necessary. They are not supported and cannot be configured in a PAS for OpenEdge instance. Because there is no Messenger, the application connection URL changes.

  In classic WebSpeed, the URL path included the Messenger in the scripts directory of the web server, and then the path to the code. For example:

  ```
  http://hostname:port/cgi/wspd_cgi.sh/. . .
  ```

  In PAS for OpenEdge, the default URL references the WebSpeed transport (`WEB`) and not the Messenger. For example:

  ```
  http://hostname:port/web/. . .
  ```