



# Progress Application Server (PAS) for OpenEdge

## Technical Migration Guide

# Introduction

The Progress Application Server (PAS) for OpenEdge is a secure, standards-based application server for OpenEdge applications, replacing the classic AppServer. It utilizes system resources very efficiently to improve scalability, and eases installation, configuration and management.

PAS for OpenEdge is an enterprise-class application server that connects your application business logic and data to a variety of client technologies, helping you bring your application into the future by modernizing user experiences and limiting security vulnerabilities. It is built upon the widely adopted Apache Tomcat® Web Server environment to provide industry-standard security via the Spring Security framework. On top of that, it also brings features such as clustering and load balancing to ultimately improve scalability and extensibility.

PAS for OpenEdge extends the Apache Tomcat Web Server to manage the specific needs of OpenEdge clients, including ABL, Open Clients (.NET and Java), WEB, REST and SOAP. PAS for OpenEdge is a crucial part of a continuous available deployment architecture, that protects applications against downtime and ensures users remain connected to their systems of record (including documents, data files and business applications). Continuous availability combines the strategies of high availability and continuous operations to address planned and unplanned outages. PAS for OpenEdge, along with a fault-tolerant deployment architecture, is designed to keep your business application running without any noticeable downtime.

This document highlights PAS for OpenEdge best practices for continuous availability with a strong focus on strategies to migrate ABL applications from classic AppServer and WebSpeed to the unified PAS for OpenEdge. Migration to PAS for OpenEdge 12.2+ is recommended.

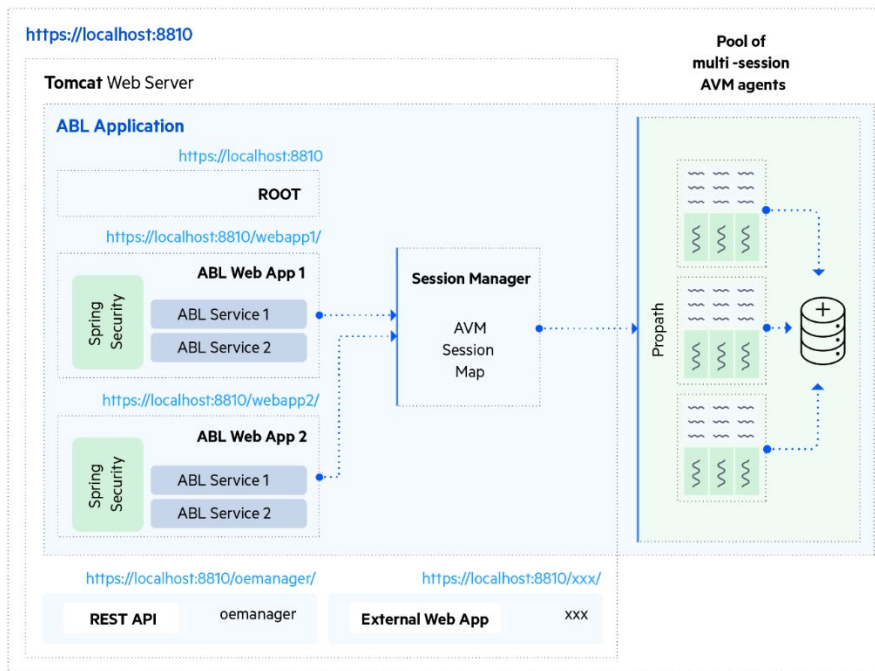
## Contents

Introduction.....	2
Understanding PAS for OpenEdge.....	3
Configuration for PAS for OpenEdge.....	9
PAS for OpenEdge Runtime.....	11
PAS for OpenEdge Management.....	13
Migration Strategy.....	15
ABL Client Migration.....	22
REST Client Migration from Classic AppServer.....	23
SOAP Client Migration from Classic AppServer.....	25
WebSpeed Migration.....	26
Open Client Migration.....	31
Server Sizing.....	33
Next Steps.....	36

# Understanding PAS for OpenEdge

## Architecture

PAS for OpenEdge is an Apache Tomcat web server that includes support for OpenEdge ABL applications. It supports a variety of clients, including ABL, browser-based clients, REST and mobile clients. Apache Tomcat is built using industry standards such as Spring Security, a powerful and highly customizable authentication and access-control framework for enterprise applications. PAS for OpenEdge is designed for easy business application deployment and simplified application management. The following diagram is a representation of a PAS for OpenEdge instance, note all components reside on a single machine:



The components in the diagram are:

- The **Tomcat Web Server** (PAS for OpenEdge Instance). Tomcat is used as both a web server to deliver static web content and as an application server for ABL applications. Tomcat accepts incoming HTTP/S requests from clients such as ABL, browser clients and mobile devices and routes those to the web applications deployed to the Tomcat web server. All clients (including ABL and OpenClient) communicate with PAS for OpenEdge using HTTP. You can find details on the Tomcat environment at [PAS for OpenEdge Instances](#).
- An **ABL application** is a grouping of ABL web applications and business logic that operate within a unique PROPATH, set of database connections, application configuration such as locale, and security configuration. An ABL application also provides perimeter security. A single ABL application can be used to replace both classic AppServer Agents and WebSpeed Messengers. You can find details on ABL applications at [ABL Applications](#).

- One or more **ABL web applications**. A web application is the next level of isolation within an ABL application. It is identified by a unique URL path and security configuration. All HTTP/S requests to that URL are mapped to a published API defined as an ABL Service or are used to access static content. Each API is a logical representation and maps to specific ABL code which permits obfuscation of sensitive implementation details such as method and parameter names. Each web application is packaged as a web application archive file (.war). You can find details on ABL web applications at [PAS for OpenEdge Web Applications](#).
- One or more **ABL Services**. ABL Services define domain- and micro-service APIs as part of an ABL web application. The ABL Service is identified as a resource in the URL and mapped to business logic within the ABL Application. The mapping details are transport type specific—one of WEB, REST, SOAP or APSV are used. You can find details on ABL services at [Create an ABL Service](#) and [ABL Service Artifacts](#).
- One or more **Multi-session Agents (MSAgents)**. An MSAgent is a specialized AVM that can run multiple ABL sessions concurrently, allowing one MSAgent to handle requests from multiple PAS for OpenEdge clients. An MSAgent maps one-to-one with an OS process. Since you can run multiple ABL sessions within a single process, this highly scalable architecture uses less system resources than classic AppServer. When an MSAgent starts up, it runs the Agent Startup procedure if one was specified. This procedure can be used to perform tasks required by all server sessions that run in the agent. One common task is to create all self-service database connections that are shared by the server sessions created and managed by a given MSAgent. Each MSAgent is a user with its own self-service connection to a given database, and all its server sessions share that same connection as separate users. You can find details on Agents and ABL Sessions at [Agents and Sessions](#) and [ABL Sessions](#).
- The **Session Manager**. The Session Manager is responsible for processing all incoming requests and routes them to the ABL Sessions within a MSAgent. It also manages the pool of ABL Sessions that can be running in one or more MSAs. You can find details on the Session Manager at [Broker Agent Architectures](#) and on the session pool at [ABL Session Manager and Session Pool](#).
- **PAS for OpenEdge Clients** send requests to a PAS for OpenEdge instance. PAS for OpenEdge Clients can be an ABL Client, Java Open Client, .NET Open Client, browser Client, REST clients, SOAP clients, Web UI and Mobile UI. In general, PAS for OpenEdge expects clients to operate in a stateless manner. For the purposes of supporting classic AppServer applications, the APSV and SOAP transports give you a means to emulate session-managed and session-free models. ABL Services identify the type of clients that can be supported by including a transport identifier in the URI of the service. Supported transports are:
  - **APSV** for ABL clients, Java, and .NET Open Clients
  - **WEB, REST** for RESTful requests
  - **SOAP** for SOAP requests
  - **Static** for Static content
 You can find details on transports at [Transports and Services](#) and details on client access migration to PAS for OpenEdge in the [Application Migration and Development Guide](#).

You can find details on the PAS for OpenEdge components described above at [What is PAS for OpenEdge?](#)



## Sessions in PAS for OpenEdge

It is important to understand the multiple types of sessions that are part of PAS for OpenEdge since PAS for OpenEdge acts as both a web application server and as a multi-session Agent. This overloaded use of the term “session” can be defined as follows:

- Client Session ([Client application](#))
- [HTTP/S Session](#)
- [Login Session](#)
- [Session Manager Session](#)
- [ABL Session](#)

### Client Session

A Client Session is implicitly created and managed by the client application or browser when a user connects to an ABL Application. This Client Session is used when the client plans to communicate a contiguous sequence of requests with PAS for OpenEdge such as login, a set of requests, and logout. The Client Session maintains the context for the connection by storing Apache Tomcat’s JSESSIONID returned on login and passing it on all subsequent requests to PAS for OpenEdge. This session is exclusively client-side and is not known to PAS for OpenEdge. You can find information about JSESSIONID at Wikipedia’s [Session ID](#).

### HTTP/S Session

An HTTP/S Session is created and managed by Apache Tomcat, as the default method for preserving session and contextual information when a login request is received from a client. This session captures execution state and context across a contiguous sequence of HTTP/S message exchanges from a specific client by assigning a unique identifier, JSESSIONID. The JSESSIONID is a cookie generated by Servlet containers like Tomcat or Jetty and used for session management for HTTP/S protocol.

HTTP/S sessions are an enabling technology for load balancing. With HTTP/S sessions, session information can be shared between a cluster of PAS for OpenEdge instances. To add an instance to a cluster, you must turn on the cluster property in the `/conf/server.xml` file of the instance.

OpenEdge 12.2 and later supports Java 11 which provides support for TLS 1.3. The default is still TLS 1.2. The cert store has not changed, neither has the implementation of how we use certificates. You can find details at [Manage Certificate Store Files](#)

Note: HTTP Sessions can be used for PAS for OpenEdge transports REST, WEB and static according to the login session model defined in Spring security. The APSV transport uses HTTP Sessions by default but you can opt out in the `openedge.properties` file and the SOAP transport does not use HTTP Sessions.

### **Session Manager Session**

The Session Manager creates its own Session in response to a request from a PAS for OpenEdge client to manage the execution of a client request to an ABL Session. Each Session Manager Session maps one-to-one to an HTTP/S Session as defined above. Session Manager Sessions exist for the life of the request except for the APSV transport where the session exists from the initial client login until the client disconnects. These sessions are internal and never exposed to the developer/user.

### **Login Session**

A Login Session manages a user identity with a unique id-token issued by a trusted Authentication Provider. This id-token is used by a web application to authorize access to file and application resources. A Login Session may exist for a single web application that employs its own private Authentication Provider, or it may exist for multiple ABL web applications that span multiple PAS for OpenEdge instances.

It is common for the ABL web application to produce a Login Session containing a Client-Principal id-token. The Client-Principal id-token is available to the ABL application for authentication (of database connections) and authorization processes. You can find details on the Client-Principal at [Introduction to OpenEdge security domains](#).

When running in an SSO enterprise environment, the PAS for OpenEdge client creates a Login Session with an external Authentication Provider id-token. The PAS for OpenEdge client sends this id-token to the web application's Spring Security component where built-in Spring SSO Authentication Providers (i.e. OAUTH2, SAML2) validate the id-token on each HTTP/S request. If valid, PAS for OpenEdge will produce an equivalent Sealed Client-Principal id-token that is delivered to the ABL application for use in authentication and authorization processes. You can find details on SSO at [About single sign on support](#).

Note: SSO is available for the APSV, WEB and REST transports but not for the SOAP transport.

### **ABL Session**

An ABL Session is a concept that has existed since the beginning of OpenEdge. An ABL session is a secure, segregated environment in which ABL business logic can run. For classic AppServer one ABL Session is mapped to one AVM running in one OS process.

With PAS for OpenEdge and the MSAgent, a single OS process can contain multiple ABL Sessions to handle concurrent client requests. ABL Sessions are uniquely identified by a Session ID and contains a private copy of ABL application r-code, variables, handles/objects, database connections, network connections, and OS file-system channels. A pool of available ABL Sessions is managed by the MSAgent and made available to the Session Manager. You can find details on the behavior of ABL sessions in PAS for OpenEdge at [Operating modes and session types](#).

## Architectural differences between Classic AppServer and PAS for OpenEdge

While both PAS for OpenEdge and the classic AppServer run ABL business applications, the architecture and configuration are fundamentally different. As defined earlier, PAS for OpenEdge is built using standard components such as the Apache Tomcat web server which it extends with pre-built ABL web applications to be able to run ABL code. It is important to understand the differences between the architectural models and components of classic AppServer applications to PAS for OpenEdge so you can best migrate your business applications. You can find details on these differences at [Differences between Classic AppServer and PAS for OpenEdge](#).

The classic AppServer components AIA and the NameServer are not used by PAS for OpenEdge. The classic AppServer scheme of AppServerDC is not available in PAS for OpenEdge. Instead, all communication is done over HTTP/S using a URL for the connection. Lastly, the AdminServer is not needed by PAS for OpenEdge to service requests; but it can still be used for administration using OpenEdge Management and OpenEdge Explorer to manage a PAS for OpenEdge instance.

The rest of this section looks at the main differences and important considerations for migrating to PAS for OpenEdge. When you are ready to migrate, you can find details on the steps to move your application from classic AppServer to PAS for OpenEdge at [Quick Start: Move classic AppServer Applications to PAS for OpenEdge](#).

## Application Models

In classic AppServer, the operating modes supported for client connections are defined in `ubroker.properties` with each AppServer instance supporting only one operating mode. With PAS for OpenEdge, ABL and OpenClient clients now control the connection mode by specifying session-managed or session-free application models in the `CONNECT ()` method. Additionally, a single PAS for OpenEdge instance can support connections from both session-managed and session-free ABL clients and replicates the classic AppServer behavior in triggering the connect and disconnect event procedures. You can find details at <https://docs.progress.com/bundle/pas-for-openedge-develop-applications/page/Programming-for-a-PAS-for-OpenEdge-application-model.html> for information about programming each model.

When establishing a connection, the client defines the application model to an ABL Web application when making a connection. Once connected, requests from that client can execute in the first free ABL Session regardless of whether the connection is session-managed or session-free. To emulate the state-managed operating modes of the classic AppServer, the client can create a session-managed connection to PAS for OpenEdge, which automatically runs the specified `CONNECT` and `DISCONNECT` event procedures. ABL, OpenClients, and SOAP clients can connect using a session-managed model. You can find details on migrating Operating Modes to Application Models at [Migrate classic AppServer operating modes](#).

To emulate classic AppServer's state-aware and state-reset behavior, you also need to bind the connection in the `CONNECT` procedure, unbind and `QUIT` in the `DISCONNECT` procedure to return the ABL Session to its initialized state. You can find details on binding the connection at [Migrate classic state aware operating mode](#).

## ABL Applications

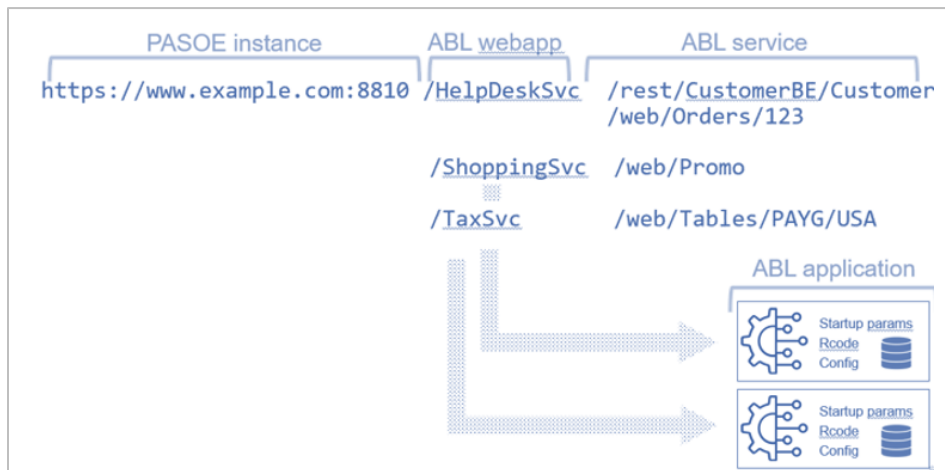
An ABL Application is a collection of ABL web applications and business logic that operate within a unique PROPATH, set of database connections, application configuration such as locale, and security configuration. ABL applications can be used to manage a group of Services.

### Operating Mode -> Application Models

In classic AppServer the operating mode was a property of the AppServer itself. With PAS for OpenEdge, this property has been renamed to *application model* and it is now the client connection that asserts the model. Most connections are session-free; only the APSV and SOAP transports can use the session-managed application model. You can find details on application models at [Migrating AppServer operating modes](#) and [Understand application models](#) and later in section [Application Models](#) this document.

### ABL Application URL Considerations

The URL to access ABL business logic is different than the URL used by classic AppServer. The PAS for OpenEdge instance is the first part of the URL, the ABL Web Application is the second and the ABL Service with optional parameters is last. Notice that the ABL application is not part of the URL. This is possible since all ABL web applications must be uniquely named for the PAS for OpenEdge instance.



### ABL Application Packaging and Deployment

ABL Applications often contain multiple ABL Services, potentially using multiple transports. These services are deployed as ABL web applications in PAS for OpenEdge using a .war file. While you can leave your code directories as is, we recommend using a modern, best practices application structure that can make ABL Applications easier to deploy, scale, and extend. This new structure works well for multiple ABL web applications and sharing behavior and configurations as desired. You can find details on these best practices at [ABL application structure](#) and [Optimize PAS for OpenEdge for continuous operations](#).



Once your application is organized as desired, you must package your application as a standard WAR file which can be generated from the command line or Progress Developer Studio for OE. You can find details on deployment at [WAR file structure](#) and [Export an ABL Web Application](#).

#### ABL Application Deployment

Starting in 12.2, ABL Applications can also be deployed as an OpenEdge Application Archive (OEAR). This specific type of zip file contains web application resources that can be deployed to a PAS for OpenEdge instance in a single operation. The package can be exported from or imported to a PAS for OpenEdge instance. You can find details at [OpenEdge Archive Structure](#).

Once your application package is built, it is ready to be deployed to a PAS for OpenEdge instance. You can find details on deployment to PAS for OpenEdge at [Deploy an OpenEdge Application Archive using "tcman"](#). The PAS for OpenEdge instance directory is self-contained and can be easily copied to duplicate or move the instance to a production environment. You can find details at [Package an instance for production](#).

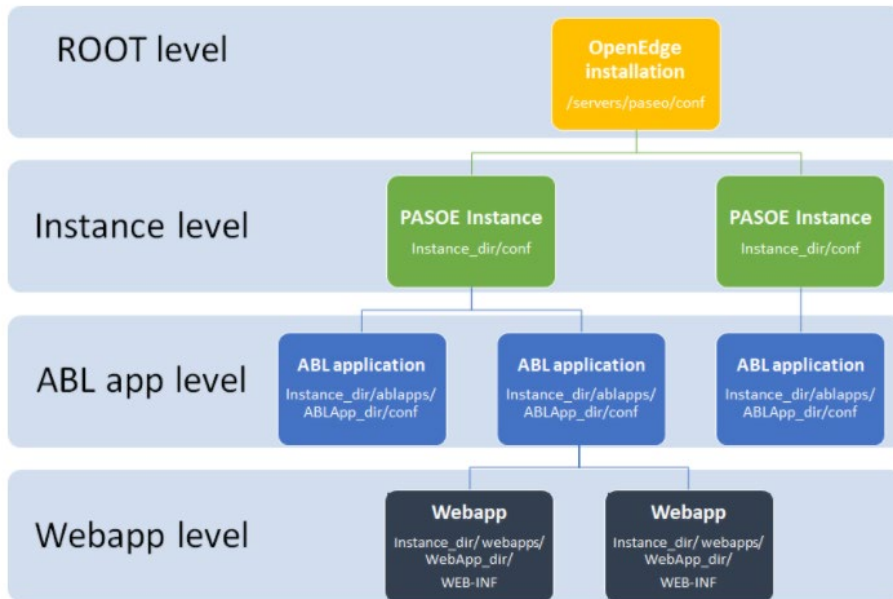
#### Application Security

You will need to migrate your authentication and authorization model. Much of your security might be handled within your classic AppServer code and will migrate directly. Consider external security processes and configuration that will need migration especially for the REST Adapter and classic WebSpeed (i.e. IIS, HTTPD). PAS for OpenEdge automatically use the Spring Security framework to perform authentication and authorization operations on all incoming requests. You can find details on Spring security at [Learn about Spring Security](#).

## Configuration for PAS for OpenEdge

PAS for OpenEdge offers flexible configuration for each instance, replacing the system-wide `$DLC/properties/ubroker.properties` configuration file used for classic AppServer with an instance specific the `instance-dir/conf/openedge.properties` file. There is no shared properties file that can be referenced by multiple PAS server instances. The PAS for OpenEdge product is in the OpenEdge installation under the `$DLC/servers/pasoe`. There you will find the shared libraries, utilities, configuration files, and default templates used to create PAS for OpenEdge instances. Configuration is file-based where the default configuration can be found in `$DLC/servers/pasoe/conf` and this is used as the starting point for all new instances.

When a new PAS for OpenEdge instance is created, the relevant artifacts from `$DLC/servers/paseo` are copied to the instance home directory `instance-dir`. The instance can then be customized for its ABL application, ABL web applications, ABL services transports, and security in the instance specific configuration files as shown below:



When migrating from classic AppServer, each named AppServer section will become its own ABL application specific `openedge.properties` file in the PAS for OpenEdge instance directory. Environment variables for the classic AppServer that were set in the `ubroker.properties` file are now set in a script file in the `instance-dir/bin` directory with a `_setenv` suffix. You can find the list of configuration files at [Configuration and properties files](#) and you can find details on the PAS for OpenEdge Instance structure at [Instance directories](#).

There are several tools available to for configuration including command line scripts, REST APIs, OpenEdge Management and OpenEdge Explorer. Your classic AppServer configuration can be converted to the new `openedge.properties` format using the `$DLC/bin/paspropconv` tool which maps classic properties into the appropriate PAS for OpenEdge properties. You can find the list of available tools at [PAS for OpenEdge configuration tools](#).

The major configuration files for a PAS for OpenEdge instance are in the `instance-dir/conf` directory:

<code>openedge.properties</code>	Application specific properties similar to <code>ubroker.properties</code> , and some PAS-specific properties. Edits are always needed here and can be done using <a href="#">OEPROP</a> or OpenEdge Management.
<code>catalina.properties</code>	Tomcat specific properties used by the webserver. Small edits such as port values are normally needed here using <a href="#">PASMANT</a> for system-wide commands or <a href="#">TCMAN</a> if you are in the instance directory. See <code>instance-dir/conf/catalina.properties.README</code> for more information.
<code>jvm.properties</code>	Contains a list of the Java JVM startup command line options. Small edits such as JVM memory are normally needed here using your favorite text editor.
<code>appserver.properties</code>	Specifies Java properties used by PAS for OpenEdge and ABL web applications. Edits are not normally needed here but if done use <code>tcmn</code> . See <code>instance-dir/conf/appserver.properties.README</code> for more information.

You can find details on configuring your instance at [Migrate classic AppServer properties](#) and the property migration utility at [PASPROPConv](#).

Note: Rather than a one-to-one mapping from classic AppServer to PAS for OpenEdge, tuning parameters for min/max/initial agents should be tested and adjusted as necessary, particularly for connections/sessions per agent within PAS for OpenEdge. You can find details at [Modify environment variables](#).

## PAS for OpenEdge Runtime

### PAS for OpenEdge Transports

As stated earlier, PAS for OpenEdge replaces the classic AppServer, WebSpeed, REST Adapter, and the Web Services Adapter (WSA). To support these different protocols, HTTP/S requests sent to PAS for OpenEdge has a transport identifier as a key component of the URL. The transports available are:

Transport	URL Path	Notes
APSV	<code>/apsv</code>	Supports the OpenEdge AppServer protocol (binary) over HTTP/S
REST	<code>/rest</code>	Supports REST RPC using a <code>.paar</code> file; replaces REST Adapter
SOAP	<code>/soap</code>	Supports SOAP 1.1; replaces WSA
WEB	<code>/web</code>	Supports RESTful APIs and compatibility for classic WebSpeed applications; replaces classic WebSpeed

For production environments, you must enable the supported transports for the instance in the `instance-dir/conf/openedge.properties` file since these are all disabled by default. You can find details on transports at [Transport URLs](#) and their properties at [Managing REST transports](#), [Managing SOAP transports](#), [Managing APSV transports](#), and [Managing WEB transports](#).

## Installation Options

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

- A **development mode** product: Progress Development Application Server for OpenEdge (Progress Dev AS for OE in the licensing) is configured as a web server for developing and testing OpenEdge applications.
- A **production mode** product: Progress Production Application Server for OpenEdge (Progress Prod AS for OE in the licensing) is configured as a secure web server for OpenEdge application deployment. This product enforces security best practices. You must make several configuration changes before you can run and deploy your applications to a production instance.
- A **limited production mode** product: Progress Application Server for OpenEdge Lite (PAS for OE Lite in the licensing) is configured as a secure web server for OpenEdge application deployment. Again, configuration changes are required.

The difference between a development and production PAS for OpenEdge is mainly a matter of security configuration. A development PAS for OpenEdge instance allows unrestricted access by a user or a group of users and is appropriate for initial migration and development efforts with PAS for OpenEdge. A production PAS for OpenEdge instance restricts access to everyone except authorized users and limits control to system administrators and is useful for testing, staging and production. Performance and load testing should always be done on a production PAS for OpenEdge where resource sharing is limited. Additionally, there is a hybrid mode where you can install both development mode and production mode in a single OpenEdge installation. This hybrid installation allows you to create a PAS for OpenEdge instance with a development mode configuration and the load testing capabilities of production mode.

You can find details on PAS for OpenEdge product modes at [About development and production instances](#) and product install differences at [Development server and production server security issues](#).

## PAS for OpenEdge Instances

PAS for OpenEdge is a Web Application server based on Apache Tomcat® and the OpenEdge MSAgent. For this purpose, PAS for OpenEdge supports ABL web applications accessed by means of HTTP-specific request-response programming model and the MSAgent runs ABL code.

PAS for OpenEdge comes with a default ABL application containing an empty ABL web application (`oeabl.war`). This web application controls the security access normally based on the licensing modes in the previous section. Once your PAS for OpenEdge instance is created, you will add your ABL code, ABL services, and perform additional configuration for your deployment needs.

The -Z parameter identifies the security model when a new PAS for OpenEdge instance is created. The configuration set per -Z value affects the ROOT web application as follows:

-Z value	Webapps deployed by default	ABL Services enabled	Default ABL Security model	Use case
Dev	oeabl.war as ROOT	All transports are enabled. Default WebHandler supports WebSpeed requests	Anonymous	Development
Prod	oeabl.war as ROOT	All transports are disabled. Default WebHandler rejects all requests: <ul style="list-style-type: none"> <li>• 405/Method Not Allowed</li> <li>• or 501/Not Implemented</li> </ul>	Anonymous	Production
Pas	noaccess.war as ROOT	N/A	N/A	Build pipeline

It is the responsibility of the administrator to update the security model according to application requirements. If a -Z parameter is not specified, the value of the web server `appserver.properties:psc.as.security.model` defaults to the license mode. You can find details on -Z at [About security models](#).

You can replace this ROOT application at your discretion but note that Tomcat requires that there always be a ROOT application. You can find more details at [The ROOT application](#).

You can also deploy additional ABL web applications as described in the [Deployment Architecture and Configuration](#) section later in this document. You can deploy the management web applications, `manager.war` and `oemanager.war`, during the creation of development instances using the `-f` option.

## PAS for OpenEdge Management

### Managing a PAS for OpenEdge Instance

The OpenEdge Manager web application (`oemanager.war`) manages PAS for OpenEdge instances through a REST API for remote administration of the ABL web applications and MSAgent. You can find more details about the `oemanager` web app at [Manager Applications](#).

The main 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. You can find details on `TCMAN` at [Web application management with TCMAN](#) and [The tcman command](#).

You can perform the same management operations using OpenEdge Management and OpenEdge Explorer. You can find details at [Configure a PAS for OpenEdge instance with OpenEdge Management](#).



A new feature in OpenEdge 12 is the PAS for OpenEdge HealthScanner. It can detect potential problems with a PAS for OpenEdge instance so that the instance can be taken out of service before a failure occurs. The HealthScanner continuously monitors key system health metrics, such as CPU health, REST ping health, HTTP request health, and disk space and memory health, and generates an overall score. If the score falls below a certain threshold, a server instance can be taken out of service by an elastic load balancer and replaced before any disruption of service is incurred. You can find details at [Use the OpenEdge HealthScanner](#).

### Application Tracing

OpenEdge Management and OpenEdge Explorer can also be used to monitor PAS for OpenEdge server performance and display performance statistics. You can find details at [Learn about PAS for OpenEdge and OpenEdge Management](#).

Deferred logging is a feature of PAS for OpenEdge to record stack trace information immediately preceding a multi-session agent crash. It also can be used to run an on-demand monitoring check on an instance through an API. Deferred logging keeps a separate memory buffer from the regular agent logging. By using a separate memory buffer, deferred logging can help keep the regular agent log to a minimal size, while providing another type of logging for reconstructing agent crashes or monitoring the application without impacting performance. You can find details on deferred logging at [Use deferred logging in PAS for OpenEdge](#).

### Deployment Architecture and Configuration

ABL Applications in PAS for OpenEdge often contain multiple ABL Services, potentially using multiple transports. These services are deployed as ABL web applications in PAS for OpenEdge using a .war file. While you can leave your code directories as is, we recommend using a modern, best practices application structure that can make ABL Applications easier to deploy, scale, and extend.

The OpenEdge Manager application (`oemanager.war`) manages PAS for OpenEdge instances through a REST API for remote administration of the ABL web applications and MSAgent. It duplicates the administration API supported by the JMX interface from Tomcat, but it uses JSON input/output payloads instead. You can find more details about the `oemanager` web app at [Manager Applications](#). You can find details on migrating your configuration from classic AppServer to PAS for OpenEdge at [Migrate Server Configuration and Management](#).

Additionally, PAS for OpenEdge comes preconfigured with a web application `oeabl.war` (ROOT) which is a pre-deployed ABL web application that can be used to run your ABL code simply by updating the PROPATH and configuration for the PAS for OpenEdge instance. The business logic is available through this ROOT application automatically.

All properties files should be edited through API calls or set using OpenEdge Management and OpenEdge Explorer. You can find details at [Configure a PAS for OpenEdge instance with OpenEdge Management](#).

# Migration Strategy

This section outlines a high-level approach for migration from classic AppServer or WebSpeed application to PAS for OpenEdge. There are four steps to consider achieving the best migration outcome.



## 1. Review Current Architecture

To migrate successfully, your current application should be reviewed to understand its architecture, components, and functionality. When studying an application several questions will inevitably be raised. Progress recommends that you create a diagram of your current architecture:

- Identify OpenEdge products and versions in use.
- classic AppServer and/or WebSpeed Brokers (`ubroker.properties`)
  - Application Modes: Stateless, State-free, State-Aware, State-Reset
  - Client Types: session-free, session-managed
  - Use of persistent/singleton/single-run procedures
  - Adapters: AIA, WSA, REST Adapter
- Web Servers, Load-balancers and NameServers
- Clients used in the application: GUI, Web, or API-based
- Application Databases (`conmgr.properties`)
- Identify batch processes run in addition on the same business logic and databases

Other important questions to consider:

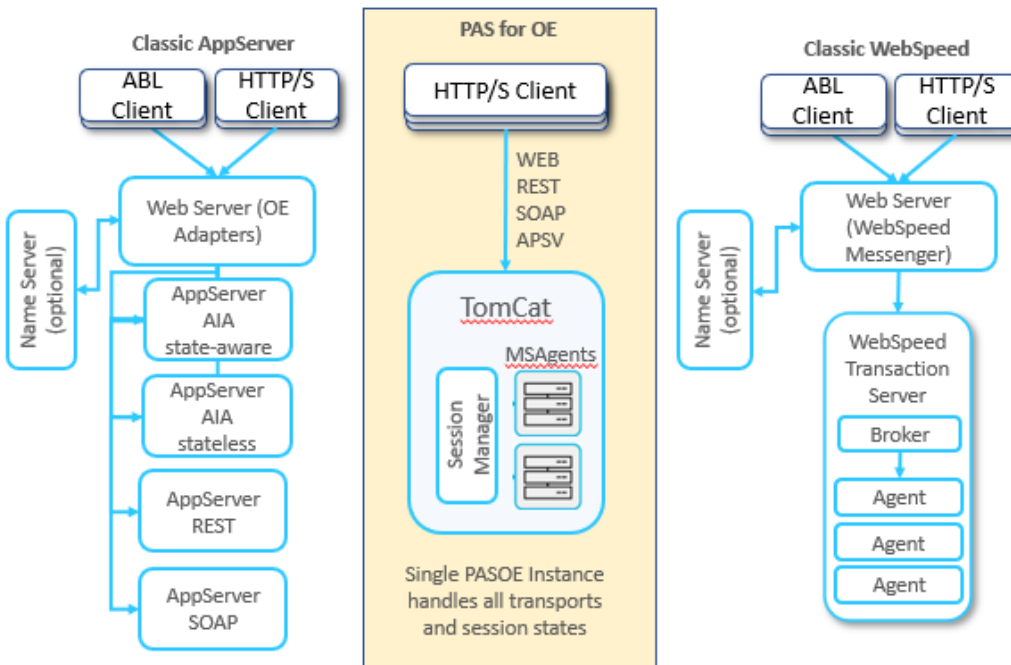
- Architecture
  - Does your application need 32-bit Windows support?
  - Are there 3<sup>rd</sup> party components that need special consideration?
- Deployment
  - Runs on-premise, in the cloud, or hybrid
  - How many servers are needed to run the application?
  - Are you running on Linux, Windows, AIX and Solaris?
  - TCP Ports, Firewall Rules
  - How is the application built and packaged?
  - Are you using a CI/CD pipeline for deployment?

With all the high-level components identified and the communication paths defined, it will be easy to visualize the components that will need to be replaced as part of the migration.

## 2. Visualize Target Architecture

Define the OpenEdge products that you will use in the new architecture. Consider Database Replication, OpenEdge Authentication Gateway, Multi-tenancy, in addition to PAS for OpenEdge. Remember PAS for OpenEdge eliminates the need for specialized adapters (AIA, WSA, REST) from classic AppServer and the WebSpeed Messenger since all communication to the business logic is managed by the transport types.

The following diagram compares the deployment architectures of classic AppServer (left) and WebSpeed (right) with the streamlined deployment architecture of PAS for OpenEdge (center).



You might want to identify new requirements for your application during the migration. Typical changes include communication mechanisms used between system components or moving to an asynchronous messaging model. It is also a good time to consider deployment environment changes including:

- Operating system change – e.g., DB on AIX, PAS on Linux
- Hardware Decisions & Control: Self-hosted or Cloud
- Advanced Topics: Clustering, Docker, Kubernetes, etc.
- Firewall rules for new ports (classic AppServer vs. PAS for OpenEdge, HTTP)
  - Moving from 1 port per classic AppServer/WebSpeed broker...
  - 1 port minimum for a single PAS for OpenEdge instance, with 1+ ABL Applications

This is a good opportunity to create a new diagram replacing classic AppServer and WebSpeed components with PAS for OpenEdge. Bear in mind that a single PAS for OpenEdge ABL Application can replace many classic AppServers and WebSpeed, since PAS for OpenEdge no longer defines the state or transport as part of the server configuration. To properly phase the migration, it is important to identify whether these new requirements are integral for the migration itself or if they can be added incrementally after the initial migration.

With a new target architecture, you can consider additional goals for modernizing your application. Common application modernizations include:

- High Availability – Uptime considerations
- Redundancy – Failover or DR options
- Scalability – Cluster or scale for load balancing
- User identity – SSO or federated identify provider
- Database – move from self-service (shared memory) to networked (client-server)

You can select one or many modernizations to consider as part of your target architecture and Progress Professional Services can assist you in this journey following a prescriptive modernization approach that includes a suggested reference architecture and identifies the high priority application imperatives and business goals for your organization. You can see more details in the OpenEdge whitepaper on [Application Evolution](#).

### 3. Plan Migration Approach

There are two migration approaches, incremental or direct. Which approach you use depends on your application complexity and system resources. It is important to become familiar with the migration steps and differences between your current and target architectures before beginning your migration. You can find details on migration at [KB Article: Migrating to OpenEdge 12](#).

#### **Incremental: Move to PAS for OpenEdge in two steps**

In the incremental approach you will do a two-step migration (classic AppServer 11.7 -> PAS for OpenEdge 11.7 and PAS for OpenEdge 11.7 -> PAS for OpenEdge 12.2). This approach allows for a more gradual learning curve as well as a side-by-side comparison of application functionality and performance on a single OpenEdge version.

In phase one, you move your application from classic AppServer or WebSpeed on 11.7 to PAS for OpenEdge on 11.7. The major considerations for this migration are:

- AdminServer is not required for operation (required if using OpenEdge Management)
- NameServer is not used, replaced by DNS and HTTP/S load-balancing
- Changing client connections (no more AppServer[DC], now HTTP)
- Your ABL, Java, and .NET clients will be operating as if they were connecting to a classic AppServer AIA adapter.
- If stateful applications are migrated, application code may need updating.

There is only one high-level step in phase one:

- Migrate application from classic AppServer or WebSpeed to PAS for OpenEdge

In this approach, the upgrade to OpenEdge 12.2 is done as a second phase. This phase requires a migration of your database and environment in addition to a small migration of your new PAS for OpenEdge ABL applications from OpenEdge 11.7 to OpenEdge 12.2. The high-level steps in phase two will include:

- DB Migration from OpenEdge 11 to OpenEdge 12
- Installing a supported JDK (Java JDK removed from install media in 12.2)
- Migrate PAS for OpenEdge 11.7 projects to 12.2; new packaging and security configurations
- Understanding new file permissions/security
- Re-compilation of code to compatible R-code

### **Direct: Move to OpenEdge 12.x and PAS for OpenEdge**

You can do a direct migration directly to OpenEdge 12.2 from 11.7 classic AppServer and WebSpeed. This is more of a big bang approach since you will migrate your applications to PAS for OpenEdge and your database and environment to OpenEdge 12.2 . A direct migration allows you to take advantage of the new online database operations, alternate database connections, automated connection failover and load balanced PAS for OpenEdge instances as well as enhanced security quickly. The high-level steps in the direct migration will include:

- DB Migration from [OpenEdge 11 to OpenEdge 12](#) or [OpenEdge 10 to OpenEdge 12](#)
- Installing a supported JDK (Note: Java JDK removed from install media in 12.2)
- Move application source and configuration to PAS for OpenEdge
- Change [client connections](#) (no more AppServer[DC], now HTTP)
- Understanding file permissions and security changes
- Re-compilation of code to compatible r-code

### **Other Considerations**

The PAS for OpenEdge architecture is designed for stateless, highly available and scalable communication. This is a very different runtime environment than classic AppServer and WebSpeed. It might be necessary to refine your development and deployment processes when moving to PAS for OpenEdge. The top areas to explore are:

- Inefficient code base
- Distributed development
- Automated cleanup scripts used due to leaks, DB locks

You can find details at [Differences between classic AppServer, WebSpeed Transaction Server and PAS for OpenEdge](#).

This is a good time to evaluate your testing strategy. Do you depend on manual and/or automated testing to validate the functionality? As you implement new system features such as scalability, consider how will you test this. It is also a good time to update/enhance your build to a continuous integration and continuous deployment (CI/CD) model. PAS for OpenEdge provides many configuration options and the build process can be set up to reuse your build and deployment tools for multiple ABL Applications.



## 4. Migrate

Migration of your applications from classic AppServer and WebSpeed to PAS for OpenEdge requires a repackaging of your code in a new way. It is important to look at the various components of the application and then determine how each component, each logical piece of the application will be migrated and the proper sequence.

ABL Applications often contain multiple ABL Services, potentially using multiple transports. These services are deployed as ABL web applications in PAS for OpenEdge using a .war file. While you can leave your code directories as is, we recommend using a modern, best practices application structure that can make ABL Applications easier to deploy, scale, and extend. This new structure works well for multiple ABL web applications and sharing behavior and configurations as desired. You can find details on these best practices at [ABL application structure](#). And [Optimize PAS for OpenEdge for continuous operations](#).

There are some good tips and additional material references at [Migrating to OpenEdge 12](#) and [PAS for OpenEdge Architecture](#) and [Design and Implementation Considerations](#).

### Migration Tools and Utilities

There is a plethora of command line tools designed to support your migration and future development. These include:

- Commands/Utilities
  - [paspropconv](#) – Harvest/convert properties from classic AS/WS (1-Time)
  - [tcman](#) for PAS for OpenEdge instance management; for example:
    - `tcman create` – Creates a new PAS for OpenEdge instance
    - `tcman deploy/undeploy` – Deploy or undeploy ABL web applications, and indirectly ABL applications
  - [tcman](#) for web application management; for example:
    - `deploy` – deploy a web application
    - `enable` – start a web application
  - [oeprop \[-f\]](#) – Modify or merge options to `openedge.properties`
  - [secprop](#) – Adjust security options to `oeablSecurity.properties`
- Create a “script” for any repeated actions
  - `ANT + PCT` is great for this
- Third-party products
  - Text Editors: `Notepad++`, `Atom`, etc.
  - File Comparators: `BeyondCompare`, `WinMerge`, etc.

- ANT Tasks
  - [Create an OpenEdge Application Archive using an Ant Build](#)
  - [Tailor an ABL Application installation using Ant Build](#)
  - [Package an ABL WebApp ANT project](#)
  - [Package REST services](#)
  - [Generate a Data Object Service Catalog file](#)

Note: This requires installation of PDSOE and an available project
- DevOps Testing
  - Deployment Processes – Ensure all artifacts get from point A to point B
  - CI/CD Pipeline – Perform essential actions without a user interface
  - Load testing – useful to identify memory leaks and identify resource contention
- Operational Testing
  - Exercising Code – Using different clients; users vs. admins
    - Authorization in place and working
  - OS/Path Changes
    - Report output, data imports, etc.

### Application Performance and Inefficient Programming Practices

It is important to understand that with multiple sessions running in a single process, the impact of each session's memory usage become compounded. The high-performance architecture of PAS for OpenEdge can and will amplify any inefficient coding practices which may include untuned queries or lingering objects and handles which can noticeably affect your runtime execution and performance. Poorly written, untuned queries, mismanaged objects and leaky code can affect your runtime execution and performance. It is important to identify and fix these situations in your application code to avoid potential scalability issues. You can find details at [Find memory leaks using ABL object tracking](#).

You can use PAS for OpenEdge Server-Side Profiling to gather and monitor run-time performance data about ABL applications running on a PAS for OpenEdge instance. You can find details at [Use Server Side ABL Performance Profiling](#).

With PAS for OpenEdge, networked database connections can have performance issues if not configured correctly. You can find details on configuration best practices for networked connections at [Performance issues between PAS for OpenEdge instance connecting from one machine to database server on another machine](#).

### Rightsizing in a Nutshell

The inevitable question is “what hardware do I need to run my application on PAS for OpenEdge?” which is greeted with an inevitable “it depends”. Though a more appropriate generalization is as follows: size your system to provide enough resources to run your application without being wasteful. To avoid coming up short on system resources at peak times it may be typical to want to over-size a server, resulting in a machine that sits mostly idle. More recently the realm of distributed applications rely on load-balancing multiple, smaller servers to satisfy increased demand only when the situation requires. The answer to this particular problem is scaling, but first we still need to establish a suitable machine to be used in a scalable cluster.

In either case there is a cost to that computing power and will likely be a larger factor to operational budget—though the first step is to achieve the best application performance on a single server first. You can find details on tuning best practices at [Performance Tuning Basics](#) and [Tune PAS for OpenEdge instances](#).

### Cloud Computing

Cloud computing providers (AWS, Azure, etc.) offer a range of virtualization families, meaning machines specially tuned by purpose. From general purpose machines to compute-, memory-, or storage-optimized classes of machines. In addition to the varied machine families are fluctuations in pricing which are constantly being updated. Depending on your needs it should be possible to find the right combination of machine specifications to appropriately support your application. You will need enough system resources to support typical operation while maintaining some amount of buffer for unexpected spikes. This includes but is not limited to CPU's (cores, multi-threading), memory, disk speed (IOPS), and network throughput.

### On premise

Building a system to self-host is also an option, whether as bare metal or virtualization within your own corporate cloud. Just like any cloud solution there needs to be a balance between just enough computing power without overcommitting processors or memory that would not be used.

### “Scale-Up or Scale-Out?”

Briefly, rightsizing is a balance between increasing resources (bigger components) vs. adding parallel resources (more components) to handle load. Factors should and will involve knowledge (familiarity), skills (comfort), and budget (cost). For the most part, IT shops are vastly familiar with scaling up by building bigger servers or adjusting vCPU/memory allotments in virtualized environments.

Once it is understood how an application should perform at peak with specific resources it should be possible to extrapolate a scaled environment. Recommendations on specific load-balancers is not covered as part of this material, though more information can be found in this section of the [PAS for OpenEdge administration guide](#).

The key process is to **test, monitor, adjust, and repeat** as necessary to find the proper balance of resources and parameters for every application. Be sure to test your application under true production load. While your code might run in a test environment for a single user, it is critical to perform load tests to identify memory leaks, open handles, etc. that can cause problems. Identifying these conditions early gives you an opportunity to mitigate these issues using PAS for OpenEdge timeouts and other configuration settings. You can find more details on sizing at [PAS for OpenEdge tuning recommendations](#).

# ABL Client Migration

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.

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.

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. A PAS for OpenEdge session-free connection operates with the same behavior as the classic AppServer running in the state-free operating mode. You can find details at [Migrate classic AppServer operating modes](#).

## ABL Client Connections

When you migrate ABL clients to PAS for OpenEdge, it is necessary to change your connection code to use a PAS for OpenEdge appropriate URL for the connection to your new ABL Web Application. A PAS for OpenEdge URL for ABL clients contains the ABL web application name and transport. This is normally provided to the `CONNECT` method on the server object handle used to access the PAS for OpenEdge instance.

The classic AppServer scheme of AppServerDC (direct connect) is not available in PAS for OpenEdge. Instead, all communication is done over HTTP/S using a URL for the connection.

The syntax and examples of supported URLs for ABL clients are:

```
Syntax:    -URL scheme://host:port//[web-app]/apsv [-sessionModel Session-free]
Example:   -URL https://localhost:8810/apsv -sessionModel Session-free
```

You can convert an ABL client connection from classic AppServer to APSV as shown below:

```
Classic AppServer:  -S port -H host -AppServer asbroker1 -ssl
ROOT Web App:      -URL https://host:port/apsv -sessionModel Session-free
Named Web App:     -URL https://host:port/custSvc/apsv -sessionModel Session-free
```

You can convert an ABL client AIA connection from classic AppServer to APSV the same way:

```
Classic AppServer:  -URL https://host:port/aia/Aia?AppService=asbroker1
                   -sessionModel Session-free
ROOT Web App:      -URL https://host:port/apsv -sessionModel Session-free
Named Web App:     -URL https://host:port/custMgt/apsv -sessionModel Session-free
```

Note: the AppService name from classic AppServer AIA connection is not used in the PAS for OpenEdge URL. The ABL web application serves the same purpose in PAS for OpenEdge URLs (to determine which ABL application is used to service requests from a client).

When PAS for OpenEdge runs behind a load balancer, it is necessary to configure the load balancer and PAS for OpenEdge to bind a client session to a particular PAS for OpenEdge instance using a sticky session. After the initial request is satisfied by a Web application, subsequent requests are routed to the same ABL Web application running on the same PAS for OpenEdge instance. You can find details on sticky sessions at [Clusters and sticky sessions](#).

The connection lifecycle for ABL clients in classic AppServer is often the entire life of the client session. With PAS for OpenEdge, a connection is not persistent, and timeouts are more likely to occur due to the nature of the underlying HTTP/S protocol. It is important to make sure your client code manages sessions by utilizing the updated `CONNECTED()` behavior available in OpenEdge 12.4 and later. In earlier PAS for OpenEdge releases, it is recommended to write a `ping.p` procedure on the server to validate whether the connection is still valid. When a connection has timed out, you will need to detect this situation and reconnect as necessary. You can find details on PAS for OpenEdge client connections at [Migrate client connections](#) and [Connect clients with new PAS for OpenEdge transports](#).

## REST Client Migration from Classic AppServer

REST clients for PAS for OpenEdge always connect using the session-free application model. When you migrate REST services from classic AppServer/REST Adapter to PAS for OpenEdge, you can use the same URL for your REST Service. To do this you simply configure your PAS for OpenEdge instance and ABL web application name, so the URLs are identical to the ones used to access the classic AppServer.

You can migrate existing REST ABL Services to the PAS for OpenEdge REST transport in one of the following ways:

1. Use existing REST service archives `-PAAR` files can be deployed as-is into a PAS for OpenEdge instance using the `deployREST` command. This will retain the same URL scheme as classic AppServer. You can find details at [Migrate REST URLs](#).
2. Create new REST services using the same or a new URL scheme. In this model, create the REST Service in Progress Developer Studio for OpenEdge using annotations (recommended) or the visual GUI mapper. You can find details on annotations at [Annotate ABL resources using the Define Service Interface wizard](#) and the GUI mapper at [REST Expose Editor](#).



3. Convert to WEB services using the built-in Progress OpenEdge Data Object Handler (DOH) or you can write your own custom WebHandler. The WEB transport uses ABL code to dispatch HTTP/S requests to the right business logic. You can annotate your source code to create a Progress Data Object (PDO) Service or define the service using a map file. More information on the WEB transport can be found in the [Data Object Services](#) section of this document.

### REST Client Connections

When using the REST transport described in Options 1 and 2 above, you can use the same URLs by naming the ABL web application the same as your rest-app-name used for classic AppServer. You can also omit (ROOT) or change the web application name to define a new URL scheme.

The syntax and examples of supported URLs for REST clients are:

**Syntax:** `https://host:port/[web-app]/rest/service-name/resource-path`

**Example:** `https://localhost:8810/samplewebapp/rest/CRMService/Customer`

You can convert a REST client from classic AppServer to REST as shown below (note: no URL change):

**Classic AppServer:** `https://host:port/CustMaint/rest/CustomerSvc/CustConnect`

**Root Web App:** `https://host:port/rest/CustomerSvc/CustConnect`

**Named Web App:** `https://host:port/CustMaint/rest/CustomerSvc/CustConnect`

You can find details on PAS for OpenEdge REST services at [Runtime architecture and data access](#) and [Develop an ABL service using the REST transport](#).

### New or Converted REST Services

In addition to the REST transport, you can also access REST services using the more flexible WEB transport. While the WEB transport is defined later in this document for classic WebSpeed migrations, you can also use the WEB transport to support RESTful APIs. You can consider a conversion from REST clients to the WEB transport or use the new model when creating new APIs. You can find details at [Data Object overview](#).

# SOAP Client Migration from Classic AppServer

You can migrate existing SOAP ABL Services to the PAS for OpenEdge SOAP transport. ABL SOAP clients for PAS for OpenEdge connect using the application model specified in the WSDL. When you migrate, it is necessary to change your connection code to use a PAS for OpenEdge appropriate URL to connect to your new ABL Web Application or ROOT. Connection URLs for SOAP clients use the SOAP transport followed by a WSDL path that includes a URL query parameter `targetURI` to specify the web service to access.

Existing SOAP services can be deployed to a PAS for OpenEdge instance, using the [deploySOAP](#) command.

## Connections using the SOAP Transport

The syntax and examples to bind an ABL client using the `CONNECT( )` method are:

```
Syntax:      -WSDL http://host:port//[oeabl-web-appl]/soap
              /wsdl?targetURI=urn:service-name
Example:     -WSDL https://localhost:8810/samplewebapp/soap
              /wsdl?targetURI=urn:CustSvc
```

You can convert the ABL client `CONNECT` parameter from WSA to SOAP as shown below:

```
WSA:         -WSDL https://host:port/wsa/wsa1/wsdl?targetURI=urn:CustSvc
ROOT Web App: -WSDL https://host:port/soap/wsdl?targetURI=urn:CustSvc
Named Web App: -WSDL https://host:port/CustMaint/soap/wsdl?targetURI=urn:CustSvc
```

You can find details on SOAP client connections at [Migrate SOAP URLs](#). You can find details on PAS for OpenEdge SOAP services at [Develop an ABL service using the SOAP transport](#) and [Manage SOAP transports](#).

# WebSpeed Migration

You migrate existing classic WebSpeed applications to the PAS for OpenEdge WEB transport. WebSpeed migration is often a straight-forward process as most classic WebSpeed source code will run once compiled against the new target version of OpenEdge and deployed on the ABL Application's PROPATH within a PAS for OpenEdge instance. When an HTTP/S request for the WEB transport is received, it is forwarded to a specialized ABL class called a WebHandler, which will process and dispatch the request to your business logic.

You can find details at [Migrate classic WebSpeed Applications](#), [What's new and different in WebSpeed on PAS for OpenEdge](#) and [Differences between WebSpeed Transaction Server and PAS for OpenEdge](#).

Note: Both *HTML with Embedded SpeedScript (.html)* and *CGI Wrapper (.p)* WebSpeed programming models are supported in PAS for OpenEdge. Special attention is needed for *HTML Mapped Web Objects* as they are **not supported** by PAS for OpenEdge and would require a conversion of the code base.

## Migration of WebSpeed components

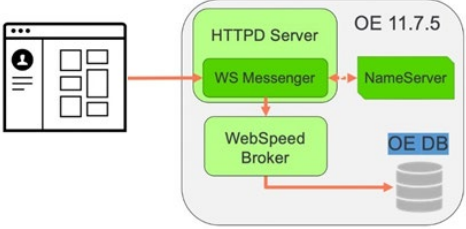
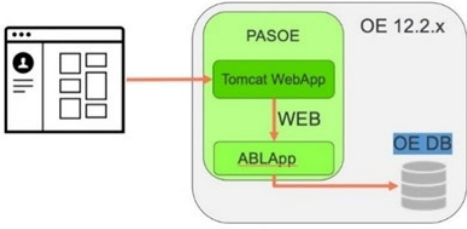
There are some significant differences between classic WebSpeed and PAS for OpenEdge. First and foremost, the WebSpeed Messenger is removed from PAS for OpenEdge as the Tomcat web server provides a direct conduit from HTTP/S web requests to the ABL runtime.

The migration process consists of moving the components of your WebSpeed application to the PAS for OpenEdge environment. The following table provides a summary of the components of a WebSpeed application and the migration required for each of them to PAS for OpenEdge:

classic WebSpeed	PAS for OpenEdge
Static Files (HTTPD Server)	<code>instance-dir/webapps/webapp-name /static/folder</code>
CGI Wrapper (.p files)	Recompile, place in PROPATH
Embedded SpeedScript (.html)	Recompile, place in PROPATH
Mapped Web Objects	Not Supported
URL (CGI script mapped to WS)	URL (HTTP/S)
<code>ubroker.properties</code>	<code>openedge.properties (paspropconv)</code>
<code>web-disp.p</code>	<code>web-handler.p</code> ; <code>OpenEdge.Web.CompatibilityHandler</code> or Custom WebHandler

## Architecture Comparison

The classic WebSpeed architecture and PAS for OpenEdge architecture is shown below:

classic WebSpeed	PAS for OpenEdge
	
<ol style="list-style-type: none"> <li>1. Requests are initially serviced by a web server for the WebSpeed Messenger.</li> <li>2. Optionally, a NameServer may be consulted to determine the appropriate WebSpeed broker.</li> <li>3. Requests are routed to the WebSpeed broker to execute business logic against a database.</li> </ol>	<ol style="list-style-type: none"> <li>1. Requests are always directed to a Web Application within the PAS for OpenEdge instance.</li> <li>2. WebSpeed requests are handled using the WEB transport along with a WebHandler.</li> <li>3. The business logic executes against a connected database.</li> </ol>

## Bootstrapping the WebSpeed Application

The bootstrapping previously done in the `web-disp.p` file for classic WebSpeed is found in `$DLC/src/web/objects/web-handler.p`. If you have customized `web-disp.p` then you may need to override the method `StartProcedure()` to execute your own instance specific version of `web-handler.p` or write a custom `WebHandler` that extends the shipped `OpenEdge.Web.CompatibilityHandler`.

## WebHandlers

All HTTP/S requests for the WEB transport are forwarded to a specialized ABL class called a `WebHandler`, which will process and dispatch the request to your business logic. There are two built-in `WebHandlers` that are mapped by default:

- The `OpenEdge.Web.CompatibilityHandler` provides compatibility with WebSpeed SpeedScript and CGI Wrapper applications. This is the default handler used in an instance in a *development* environment and must be used if support for classic WebSpeed is needed in production environments.
- The `OpenEdge.Web.DefaultWebHandler` handler returns a 405 Method Not Allowed error for requests not mapped to a specific handler. This is the default handler used in a production instance

You can also create a custom `WebHandler` that can parse and process the HTTP/S request as well as provide an HTTP/S response back to the client. This is useful for splitting apart the classic WebSpeed URL space, or for extending your classic WebSpeed application.

## Connections using the Built-in Compatibility WebHandler

For most classic WebSpeed applications you can use the built-in `OpenEdge.Web.CompatibilityHandler`. This WebHandler is analogous to the WebSpeed `web-disp.p` standard routing and will properly call your business logic as defined in your classic WebSpeed application.

The syntax and examples of supported URLs for WEB clients are:

Syntax: `https://host:port/[oeabl-web-appl]/web/service-name/service-resource`  
Example: `https://localhost:8810/samplewebapp/web/CRMService/Customer`

You can convert a WebSpeed client to WEB as shown below:

WebSpeed Script: `http://webserver/cgi-bin/sample.cgi/getCust.p`  
ROOT Web App: `https://localhost:8810/web/getCust.p`  
Named Web App: `https://localhost:8810/samplewebapp/web/getCust.p`

## Custom WebHandlers

To migrate classic WebSpeed applications with complex routing, you might need to create a new custom WebHandler class to implement the necessary customizations. A custom WebHandler class can be created by either subclassing `OpenEdge.Web.CompatibilityHandler` or by writing one from scratch that implements `Progress.Web.IWebHandler`. You will need to map the URLs to a Service to your new custom WebHandler. You can find details at [Create a WebHandler Class](#).

## ABL Services

In PAS for OpenEdge you now have the ability to group classic WebSpeed functionality as an ABL Service. An ABL service exposes the functionality contained in the business logic layer as APIs for client applications to consume. As part of your migration, you can choose to organize your WebSpeed APIs as one or more ABL Services and map a WebHandler (built-in or custom) to the URLs for the service. You can find details at [Create ABL services](#).

## Connections using a Custom WebHandler

When using a custom WebHandler you will need an additional component in the URL to properly route the incoming HTTP/s requests to the WebHandler for the WebSpeed applications with a modified `web-disp.p` will need a custom WebHandler to implement complementary customizations.

You can convert a WebSpeed client to WEB with a custom WebHandler as shown below:

WebSpeed: `http://webserver/cgi-bin/samplewebapp.cgi/getCust.p`  
ROOT Web App: `https://localhost:8810/web/CRMService/getCust.p`  
Named Web App: `https://localhost:8810/samplewebapp/web/CRMService/getCust.p`

## High-level Migration Steps

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

1. Move the application's static files to a specific folder in the PAS for OpenEdge instance or within another web server such as Apache, IIS, nginx, etc. The new location of your static files requires an update to the URLs used to access your application UI if it uses these static files.
2. Update the PROPATH for the instance to include the folders that contain the application's r-code.
3. Customize the `$DLC/src/web/objects/web-handler.p` source file to initialize your session for shared variables. In classic WebSpeed these were normally defined in the `web/objects/web-util` file. These variables include `SelfURL`, `AppURL`, `AppProgram`, `SCRIPT_NAME`, and `PATH_INFO`.
4. Determine whether you can use the built-in compatibility web handler or if you need to create a new WebHandler class. If you need a custom WebHandler, your new WebHandler class must be found in PROPATH for the application and you need to configure the PAS for OpenEdge instance properties to map URLs to your WebHandler. You can find mapping details at [Managing WEB handlers](#).
  - o When HTTP/S requests come into the WebHandler, they are dispatched to the actual business logic, in the case of migration this will be your existing code. If you have created a custom WebHandler class, you might need to edit your API to match your new dispatch methodology.
5. Enable the `WEB` transport on the instance and use the preconfigured or custom WebHandler configuration in the `openedge.properties` file.

## Supporting classic WebSpeed Client URLs

To maintain the URLs used in your classic WebSpeed applications, you can use Tomcat's URL rewrite capabilities, using its rewrite valve. To enable the rewrite valve for an ABL web application:

1. Edit the ABL Web application context file in `instance-dir/webapps/webapp-name/META-INF/context.xml`. Inside the `<Context>` element, add the following:

```
<Valve className="org.apache.catalina.valves.rewrite.RewriteValve" />
```
2. Create the file `instance-dir/webapps/webapp-name/WEB-INF/rewrite.config` to define a rewrite rule for Tomcat. Your rewrite rule can be simple or complex. The example rule below replaces `cgi-bin` with `web` and adds the next path segment (the script) as a query string named `cgi`:

```
# this replaces /cgi-bin/script/program.p with /web/program.p?cgi-script=script
RewriteRule (.*)/cgi-bin/(.*)/(.*) $1/web/$3?cgi-script=$2 [QSA]
```

You can find details on Tomcat's rewrite valve by looking at the specific version of Tomcat. For example you can find the details for Tomcat 9 at <https://tomcat.apache.org/tomcat-9.0-doc/rewrite.html>.

## Web Transport Configuration

In classic WebSpeed, you configure the WebSpeed Transaction Server and define features with environment variables and entries in the `ubroker.properties` file. In PAS for OpenEdge, this is replaced with the instance specific `instance-dir/conf/openedge.properties` file in the `ROOT.WEB` section as shown below:

```
[oepas1.ROOT.WEB]
  adapterEnabled=1
  defaultCookieDomain=
  defaultCookiePath=
  defaultHandler=OpenEdge.Web.CompatibilityHandler
  srvrDebug=0
```

You can find help information that explains these parameters and their settings

in `$(CATALINA_BASE)/conf/openedge.properties.README`

New for PAS for OpenEdge is the configuration mapping of URLs to WebHandlers. The order of the URL mapping is important since an incoming URL is matched to the first match. This means that more specific URL patterns should appear before wildcarded mapping. The mapping is part of the PAS for OpenEdge instance configuration:

- For OpenEdge 12.2+, WebHandlers are configured with the ABL web application deployed to an instance. This properties file `instance-dir/webapps/webapp-name/WEB-INF /adapters/web/service-name/service-name.handlers`. You can find details on WebHandlers at [Deploy web handler services](#).
- For prior releases, WebHandlers are defined in the file `$(CATALINA_BASE)/conf/openedge.properties`.

## Classic WebSpeed internal ABL variables and values

Customers can override the various `init-*` procedures (with the exception of `init-variables`). To do this, set the `SUPER_PROC` environment variable to a `.p` in `$(catalina_base)/bin/webspeed_setenv.[bat|sh]`; this `_setenv` file will need to be created. The use of a separate `_setenv` file is recommended.

Examples are available for [Windows](#) and [Unix](#). This `selfurl.p` program will be added to the webspeed super procedure stack. This allows the setting of the above variables to application-specific values. An example of this program is in [selfurl.p](#).

The WebSpeed ABL code allows the overriding of the URL using the `applicationURL` configuration value. If this configuration value is needed, it can be set in `conf/openedge.properties` file, as below:

The value **must** be defined in the `[AppServer.Agent.<abl-app-name>]` section.

```
[AppServer.Agent.<abl-app-name>]
applicationURL=/cgi-bin
```



To read this value:

```
define variable appUrl as character no-undo.  
appUrl = web-context:get-config-value('applicationURL').
```

## Next Steps

Once you have migrated your application to OpenEdge 12 PAS for OpenEdge, you can review the various components of the application and find areas to improve . One built-in WebHandler is described below in [Data Object Services](#).

## Open Client Migration

PAS for OpenEdge supports session-managed and session-free application models which provides OpenClient the same connection models as the classic AppServer. 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. A PAS for OpenEdge session-free connection operates with the same behavior as the classic AppServer running in the state-free operating mode. You can find details at [Migrate classic AppServer operating modes](#).

### Open Client Connections

When you migrate Java and .NET Open Clients to PAS for OpenEdge, it is necessary to change your connection code to use a PAS for OpenEdge-appropriate URL for the connection to your new ABL Web Application. This URL contains the ABL web application name and transport. This is normally passed to the `Connection` object constructor.

The syntax and examples of supported URLs for Open Clients are:

```
Syntax:    scheme://host:port//[web-app]/apsv  
Example:   https://localhost:8810/apsv
```

You can convert an ABL client connection from classic AppServer to APSV as shown below:

```
Classic AppServer:  AppServerDC://localhost:8810 -AppServer asbroker1 -ssl  
ROOT Web App:      https://host:port/apsv  
Named Web App:     https://host:port/custSvc/apsv
```

The connection lifecycle for Open Clients in classic AppServer is often the entire life of the client session. With PAS for OpenEdge, a connection is not persistent, and timeouts are more likely to occur due to the nature of the underlying HTTP/S protocol. It is important to make sure your client code manages sessions by calling a simple `ping.p` procedure on the server to validate whether the connection is still valid. When a connection has timed out, you will need to detect this situation and reconnect, as necessary. Make sure your Open Client code performs a disconnect when terminating to avoid running out of resources.

You can find details on PAS for OpenEdge client connections at [Migrate client connections](#) and [Connect clients with new PAS for OpenEdge transports](#).

## Changes & Enhancements

There are changes to highlight in OpenEdge 12.2 related to Open Clients:

- **Exception handling** has been enhanced. In the past O4GL, when an exception (error) on the AppServer took place, a simple error was sent back to the client. In 12 the exception handling has been enhanced to return the exact error that was risen on the AppServer. If you have implemented the previous exception handling, that will still work. No coding is required to take advantage of the new exception handling.
- **Object.Finalize** has been removed from some classes as the java class was deprecated and the use of finalize is no longer considered a good practice. `Object.Finalize` has been replaced with `java.lang.AutoCloseable`. Users SHOULD make use of the try-with-resource syntax to take advantage of the JVM functionality. Not doing so may result in a compiler warning. This construction ensures prompt release, avoiding resource exhaustion exceptions and errors that may otherwise occur.
- **Constructors for Open4GLEException and Open4GLEError** (and their subclasses) have been modified to accept varargs rather than array for some constructors. If the application is using any of the open client exception classes, they may continue to use them as before, or they can update their code to the newer style syntax which is more concise. Using the old syntax may result in compiler warnings.
- **ProxyGen** has been modified to generate code that uses the updated syntax. Use the OpenEdge 12.2 ProxyGen with your existing proxy projects [ `.xpxg` ] to generate new executables. You may continue to use older generated proxies with the updated 12.2 open client implementation, but it will not take advantage of the updates.
- **Packaging** for the Java Open Client has changed as Progress has been refactoring classes and jars into a more distributed model. You will need to update your Java Open Client (O4GL) classpath.

## High Level Migration Steps

### Step 1 – Generate proxy class.

Good news, you do not have to recreate your ProxyGen project file. If you have an existing \*.xpxg file all you need to do is open that from ProxyGen. Remember to adjust your compiler to use Java 11. Once you have generated the class files, you can now import them into your project. Since the method signatures have not changed you can simply remove the earlier OpenEdge generated class files. You can find details at [Generate proxies for a Java client](#) and [Generate proxies for a .NET client](#).

### Step 2 – Update both compile-time and run-time class-path

The above section refers to our re-packaging efforts at Progress which require a change to your Java class-path. The following link gives more details. [Java Open Client Runtime Package](#).

### Step 3 – Update connect string and try-with-resource syntax.

Any connection string that you are upgrading from classic to PAS will have to be converted. The only way to communicate with the AppServer (APSV) on PAS is now through HTTP/S or HTTPS. You will have to change all of your connection strings from a traditional [AppServerDC://172.31.83.251:8210] to [http://172.31.83.251:8220/apsv].

While you are making connect string modifications, verify that you have enclosed the appObject and connectObject in try-with-resource syntax. This would be a good time to review GC and other expensive constructs such as string concatenation or auto-boxing of integers.

Note: OpenClient 12.x is backward compatible to an 11.x classic or PAS AppServer. If you are going to maintain a mix of classic AppServer (OE 11.x) and PAS for OpenEdge (OE 11.x, 12.x) only the connections to the PAS for OpenEdge need to be changed.

### Step 4 – Deploy and test applications.

At this point you should be able to deploy your application to a test environment and run some tests to verify your code is functioning correctly.

## Server Sizing

### Configuration

Tuning should begin at the Tomcat level, ensuring enough threads (read: executor threads) are available for the total concurrent requests to the PAS for OpenEdge instance. After that, tuning should move to the ABL Application level where each ABL Application is tuned for the Web Apps it must support. The specifics of this process can be found in the “[PAS for OpenEdge Tuning Recommendations](#)” section of the administration guide. This illustrates the exact parameters to alter in the `openedge.properties` file as well as how to modify these values using the `oeprop` command via the command line. You can find details on available configuration options at [Goals and Common Steps for Tuning PAS for OpenEdge Instances](#) which will address items such as the max sessions vs. max connections as well as the initial sessions value.

For a quick reference and starting point, you can begin your application testing using some of the recommendations below along with the default values for the Tomcat server in `openedge.properties`. Based on extensive testing on AWS the main differences here vs. the documentation links above are the recommendations to start no more than 20% of your sessions initially and to increase the new `minAvailableSessions` which was added in OpenEdge 12.2:

Property	Description and value
<code>maxAgents=2</code>	Keep a maximum of 2 agents (default in OpenEdge 12.2) Maximum number of multi-session agents to start Instances
<code>minAgents=2</code>	Starts a new agent if availability is below this value
<code>numInitialAgents=2</code>	Start 2 initially to satisfy the min/max configured Initial number of multi-session agents to start
<code>agentStartLimit=1</code>	Starts only 1 agent at a time if below <code>minAgents</code>
<code>maxABLSessionsPerAgent=20</code>	Default is 200 in OpenEdge 12.2. Maximum ABL sessions per multi-session agent
<code>maxConnectionsPerAgent=20</code>	
<code>numInitialSessions=4 - Start max 20% of maxConnectionsPerAgent</code>	Default is 200 in OpenEdge 12.2 Maximum connections per multi-session agent
<code>minAvailableABLSessions=3</code>	Set to 2 or 3 (new in OpenEdge 12.2, default is 1)

For best performance, you must determine the optimal settings for the PAS for OpenEdge properties that control the number of multi-session agents, sessions, and client connections that a session manager manages. In general, you must consider the number of clients, the design of your application (including the application model), and the hardware resources that run your PAS for OpenEdge instance.

The above recommendations are primarily for the use of stateless transports such as SOAP, REST, and Web where the number of ABL Sessions per agent will equal the number of Connections per agent. When utilizing the APSV transport it is important to remember that it is effectively an emulation of a traditionally stateful connection over a stateless protocol. Therefore, it may be necessary to greatly increase the value for the `maxConnectionsPerAgent` if bound client connections are to be expected. These would be AppServer requests which utilize persistent procedures or otherwise require keeping a client connection active even after executing ABL code.

Be sure to compare your results after adjusting any parameters and make further adjustments as necessary. It is advised to change one parameter at a time to accurately gauge its effect in the results.

#### Other considerations

- For a session-managed application — You must have one ABL session for each client that connects to the PAS for OpenEdge instance. That is, you need as many ABL sessions as clients that connect concurrently to the PAS for OpenEdge instance.
- For a session-free application — You can have one session work on requests for multiple clients. At a minimum, Progress recommends one server session for each CPU in the machine. You can expect each multi-session agent to handle requests from up to 20 PAS for OpenEdge clients (or more). This is your per-

agent client load. You can then scale up the number of agents to support the required multiple of per-session client load to match your total PAS for OpenEdge client load. The per-session client load also depends on the length of time it takes to complete an average client request. The longer it takes for a session to complete session requests, the fewer sessions are supported by each session, and the more sessions you need to handle the total PAS for OpenEdge client load. Thus, the total number of clients required is very application specific.

You can find more information at [Tune PAS for OpenEdge instances](#).

## Performance Testing

### Proper Sizing Based on Performance Tests

- Load Testing
  - Performance Testing – Ensure code is responding in a timely manner
    - Memory leaks (eg. ABLObjects)
    - Code profiling (inefficiencies, repeated calls)
    - External shared library access
    - File and/or process contention
  - Capacity Testing – Preparing for concurrent and sustained requests
    - The default configs are always wrong for your specific needs!
    - Tuning agent/session parameters; min/max limits; initial values
    - Adjusting timeouts

For over 15 years the “ATM Test” has been used as a benchmark utility for demonstrating throughput against an OpenEdge database. In the “[PAS for OpenEdge Machine Sizing Guide](#)” (related: [KB84208](#)) the ATM benchmark was adapted for PAS for OpenEdge and executed on various families of AWS machines. As part of this guide, the response times of the tests were monitored and compared to load on both the CPU and memory resources for the machines. The takeaway was that as machine size (read: as CPU count increased) more concurrent clients were serviced with lower response times. This was a clear illustration of the process which ran distinct and repeatable **tests, monitored** the results, made **adjustments** as needed, and **repeated** the process until a suitable response time was achieved.

The tests themselves were easily accomplished by use of JMeter to automate requests, and the “top” command on Linux was used to monitor the performance of the OS image. The section “[Tuning Basics](#)” (related: [KB84190](#)) outlines the processes for identifying bottlenecks with regards to the ATM Test. A series of factors are outlined which illustrate how either CPU or memory constraints can affect the application and how to identify the root cause based on the bottleneck symptoms.

In addition to the server resources were the balance of agents + sessions for each ABL Application which could be identified through repeated testing. Again, the key takeaway is found in the “[Tuning Parameters](#)” section of that same guide: “we advise running as many connections per agent all on a single agent with as many sessions as it can handle before performance degrades. This generally gives the best performance and the best usage of resources.”

## Performance Tooling

The following tools provide metrics which you can use to tune your PAS for OpenEdge configuration settings.

- [OpenEdge HealthScanner](#) – Useful for examining both OS and application performance in terms of a weighted average either overall or for specific metrics.
- [Server-Side ABL Profiler](#) – Can help to pin-point inefficiencies in code which could be contributing to slow-running processes.
- [OEJMX](#) and [OEM API's](#) – Both achieve the same goals but by different means. Useful for enabling metrics at runtime which can help to track memory consumption or identify potential memory leaks. Comparing request information by tracking ABL performance can be compared to Tomcat requests to identify any discrepancies for response times.
- [Pulse Metrics Diagnostics](#) – Demonstrated as part of a [TechTalk](#), this utilizes debug features added in OpenEdge 12.2 to perform automated collection of metrics from a PAS instance and collect them into a central location. This can also work with older versions of OpenEdge but requires more work and is not recommended for non-development environments.

## Next Steps

The information in this guide is meant to start your journey of migrating your business applications to PAS for OpenEdge. In addition to the references found throughout this document, Progress offers self-paced learning and reference material to continue your migration journey:

- [Next steps with PAS for OpenEdge](#)
- [Work with PAS for OpenEdge](#)
- [PAS for OpenEdge: Machine Sizing Guide](#)
- [Manage PAS for OpenEdge](#)
- Migrating classic AppServer Applications to PAS for OpenEdge
  - [Learn About Migrating Classic AppServer Applications to PAS for OpenEdge](#)
  - [Comparing the architecture of the OpenEdge AppServer and PAS for OpenEdge](#)
  - [Comparing PAS for OpenEdge to the OpenEdge AppServer](#)
- Video Series
  - [Migrate classic AppServer Applications to PAS for OpenEdge](#)
  - [Migrate classic AppServer REST Services to PAS for OpenEdge](#)
  - [Migrate WebSpeed applications to PAS for OpenEdge](#)
  - [Deploy ABL, SOAP, and REST Applications to PAS for OpenEdge](#)

You can also leverage Progress Professional Services and OpenEdge Education to assist with your migration efforts.

## Progress Professional Services

Progress Professional Services are available to help with your migration to PAS for OpenEdge. Whether you need improved security or compliance, 3rd party integration, an API first strategy, scalability, cloud migration or high availability, PAS for OpenEdge is an enabling technology that will support your initiatives.

For those working with monolithic applications, PAS for OpenEdge facilitates the effort of evolving and modernizing your application. This datasheet provides common examples of what is included with the JumpStart offering. However, the actual engagement may be customized to meet your specific business needs.

Click the button below to download the whitepaper:



## Progress Education

Let Progress help you master the latest techniques for simplifying and streamlining the development, integration and management of global enterprise business applications with PAS for OpenEdge.

- [Progress Application Server for OpenEdge Administration](#)
- [Introduction to Progress Application Server OpenEdge for Developers](#)
- [Providing Progress OpenEdge Applications as REST Web Applications](#)
- [Building REST services with Webhandlers on PASOE](#)
- [Progress Application Server for OpenEdge Developers](#)
- [Progress Application Server for OpenEdge Administration \(PASOE\)](#)

## Additional Migration Documentation Resources

### Architecture

- [Application Migration and Development Guide](#)
- [PAS for OpenEdge Architecture](#)
- [Design and Implementation Considerations](#)
- [Migrating AppServer operating modes](#) and [Understand application models](#)

### Migration

- [Migrating to OpenEdge 12](#)
- [DB Migration: OpenEdge 11 to OpenEdge 12](#)
- [Migrate client connections](#)
- [Migrate classic AppServer properties](#)
- [Connect clients with new PAS for OpenEdge transports.](#)

### WebSpeed Specific Migration

- [Migrate Classic WebSpeed Applications](#)
- [Overview of WebSpeed support in PAS for OpenEdge](#)
- [Deploy ABL WebApp project with WebSpeed functionality](#)



- [Learn About Migrating Classic AppServer Applications to PAS for OpenEdge](#)
- [Move classic AppServer code to PAS for OpenEdge](#)
- [KB: How to migrate classic WebSpeed application to PAS for OpenEdge](#)

## Management

- [Optimize PAS for OpenEdge for continuous operations](#)
- [PAS for OpenEdge configuration tools](#)
- [Configuration and properties files](#)

## About Progress

Progress (NASDAQ: PRGS) offers the leading platform for developing and deploying strategic business applications. We enable customers and partners to deliver modern, high-impact digital experiences with a fraction of the effort, time and cost. Progress offers powerful tools for easily building adaptive user experiences across any type of device or touchpoint, the flexibility of a cloud-native app dev platform to deliver modern apps, leading data connectivity technology, web content management, business rules, secure file transfer, network monitoring, plus award-winning machine learning that enables cognitive capabilities to be a part of any application. Over 1,700 independent software vendors, 100,000 enterprise customers, and two million developers rely on Progress to power their applications. Learn about Progress at [www.progress.com](http://www.progress.com) or +1-800-477-6473.

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