Progress OpenEdge

Progress Application Server (PAS) for OpenEdge

Technical Migration Guide

GUIDE

Table of Contents

Introduction

The **Progress Application Server for OpenEdge (PAS for OpenEdge)** serves as a secure, standards-based application server designed to replace the classic AppServer for OpenEdge applications. 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 integrates application business logic and data with various client technologies, facilitating the modernization of user experiences and enhancing security measures. 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, which protects applications against downtime and ensures users remain connected to their systems of record (including documents, data files and business applications).

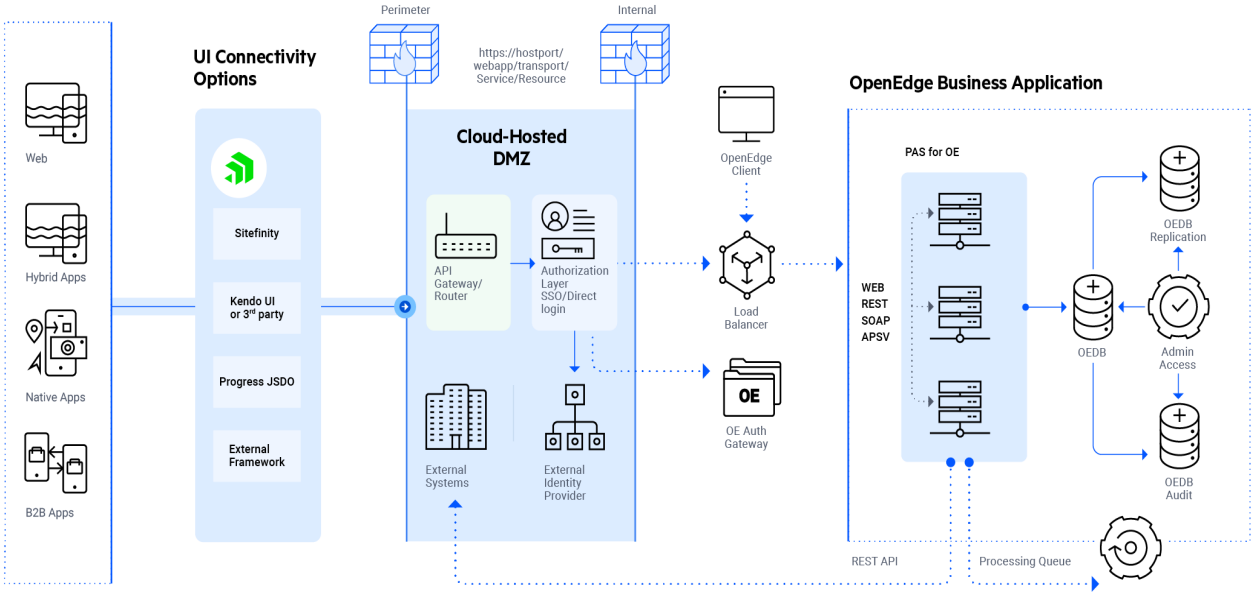
Continuous availability merges high availability and continuous operations strategies to effectively manage both planned and unplanned outages. PAS for OpenEdge, along with a fault-tolerant deployment architecture, is designed to keep your business application running without any noticeable downtime.

This document outlines best practices for ensuring continuous availability with PAS for OpenEdge, emphasizing strategies for migrating ABL applications from the classic AppServer and WebSpeed to the unified PAS for OpenEdge. Migration to PAS for OpenEdge 12.8 or later is recommended.

Understanding PAS for OpenEdge

Architecture

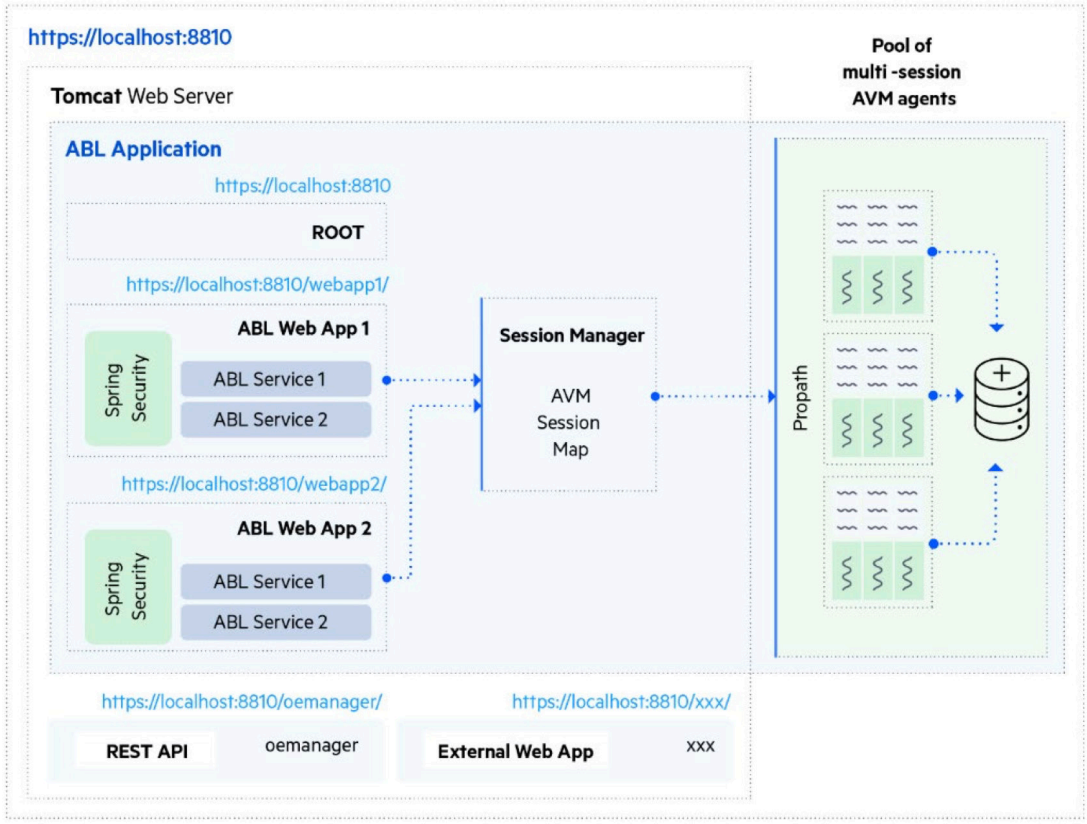
PAS for OpenEdge is an integral component of the OpenEdge environment. The following figure illustrates PAS for OpenEdge and its associated OpenEdge components.instal.



Within an enterprise business application, PAS for OpenEdge is the application server component of PAS for OpenEdge interacts with the OpenEdge database and serves client requests using transports over HTTP/S.

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: What helps to examine the customer needs and tailor the PoC:

- **Tomcat Web Server (PAS for OpenEdge Instance)**. Tomcat is used both as 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.

- One or more **ABL applications**. An ABL Application is a grouping of ABL web applications and business logic that operates 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 for 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. A 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 are 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 an internal component responsible for processing all incoming requests and routes them to the ABL Sessions within an MSAgent. It also manages the pool of ABL Sessions that can be run in one or more MSAgents. You can find details on the Session and 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 state-free requests
- SOAP for SOAP requests
- Static content (ROOT)

You can find more details on transports at Transports and Services and details on client access migration to PAS for OpenEdge in the ABL Application Code Migration.

You can find additional 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 relevant 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. You can find client interaction details at PAS for OpenEdge and Client Interaction.

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 communication and 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 12.8 and later supports Java 17 which provides support for TLS 1.3. The default is still TLS 1.2. The cert store has not changed, nor has the implementation of how we use certificates. You can find details at Manage Certificate Store Files.

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 files 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 contain 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 operating modes at Migrate classic AppServer operating modes.

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, WebSpeed Transaction Server, 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 remote 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 Move Classic AppServer Applications to PAS for OpenEdge.

Connection and Operating Modes

It is important to understand the connection and operating modes of Classic AppServer and call out any differences from that behavior in PAS for OpenEdge. In classic AppServer you had:

Connection modes: Session-managed, Session-free

Operating modes: State-aware, State-reset, Stateless, State-free

For session-managed applications or a session-free that use bound connections (State-aware, State-reset, Stateless, State-free) to support complex operations, the operations can perform fine-grained procedures and functions because the ultimate result for a client does not have to be completed in a single request since each request will go to the same ABL Session. Each request executed in a series of session-managed requests can help to assemble the context of a single operation until a single committing call completes the transaction after this context is complete.

In the session-managed connection mode, Connect and Disconnect procedures are run and establish the initial client connection state. In session-managed connections, asynchronous requests are queued and executed in the order they were run (sequential).

For a session-free application or a session-managed application that uses unbound connections (Stateless, State-free), whether simple or complex (powerful), each request runs as a self-contained operation and must complete their function in one request, since additional requests can be sent to a different ABL session. Client session context can be maintained for a session-free or for an unbound session-managed application between requests in a few ways:

- **Client management:** Use input-output parameters or HTTP cookies to pass client session context between requests from the connected client. In this case, the client session state is maintained and managed by the client.

- **Server management:** Initialize and update the client session context in the deactivate procedure. This client context is available through the SESSION:SERVER-CONNECTION-CONTEXT attribute. The session manager ensures that the context data is made available to the ABL Session where the client request is executed.

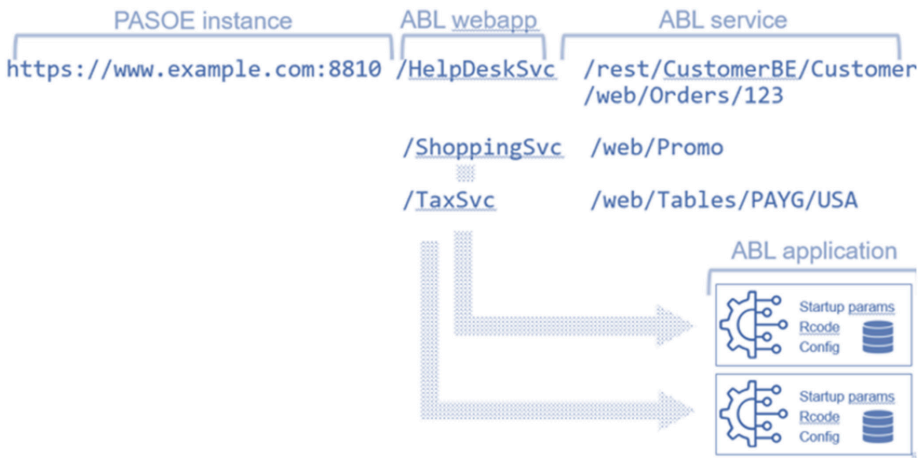
In the session-free connection mode, Connect and Disconnect procedures are NOT run. Use the activate procedure to initialize or retrieve a client session context. In session-free connections, asynchronous requests are distributed to multiple ABL sessions and executed in non-deterministic order (concurrent).

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.

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 more than one transport. 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 can package your application as a standard WAR file which can be generated from the command line or within Progress Developer Studio for OE. You can find details on deployment at Export an ABL Web Application.

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 on OEAR packaging at OpenEdge Application Archive Structure.

And lastly, you can create a ZIP file that includes both the r-code and the deployed artifacts so that the files can be extracted and registered on a production server. You can find details on ZIP packaging 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 uses 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.

PAS for OpenEdge Deployment Options

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) 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) 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) 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.

ABL Application Deployment

Once your application package is built, it is ready to be deployed to one or more PAS for OpenEdge instances. The PAS for OpenEdge instance directory is self-contained and can be easily duplicated or moved from staging to a production environment. You can find details on deployment to PAS for OpenEdge at Deploy an OpenEdge Application Archive using tcman import.

Advanced Deployment Options

PAS for OpenEdge is available as a container that can be used to deploy ABL applications using Docker and Kubernetes. Container technology in Docker and Kubernetes provides a more efficient model of deployment with better resource utilization than virtual machines. Orchestration using Docker Swarm and Kubernetes can be used to run multiple PAS for OpenEdge containers providing high availability and scalability.

Container images for PAS for OpenEdge (12.2, 12.7, 12.8) are available on Docker Hub and Progress ESD. PAS for OpenEdge is pre-installed and configured for HTTPS access in the container. You can use the PAS for OpenEdge container in a CI/CD pipeline.

You can build an image for your application that can be deployed with PAS for OpenEdge using the sidecar container pattern. Alternatively, you can build a single custom image with PAS for OpenEdge and your application together. Using the same image(s) through the pipeline stages (build, test, deploy) allows for repeatable, consistent results everywhere that it is deployed. You can build a new image and run it through the CI/CD pipeline and use a deployment strategy (for example: rolling updates) to update/upgrade your application. You can deploy your application as a container on-premises and on the cloud.

Overall, deploying ABL applications using container technology allows for faster and easier deployments and upgrades.

Related Resources:

- Learn about PAS for OpenEdge in a Docker container
- https://hub.docker.com/r/progresssoftware/prgs-pasoe

Using Archive Libraries with an ABL Application

OpenEdge 12.7 introduced archive libraries (`.apl` file extension), a collection of r-code similar to procedure libraries (`.pl` file extension), with support for signing and validation of the r-code. You can use archive libraries to organize the r-code of an ABL application and improve runtime performance and enhance the security of the application by ensuring the r-code in the library has not been corrupted or compromised.

You can use the PROPACK utility to create an archive library and optionally add attributes (signature policy, validation policy, vendor, version, and custom attributes). The attributes are written to a manifest file. The signature policy and validation policy can be set to only allow r-code in archive files that have been signed.

You can use the PROSIGN utility to sign and verify the archive library. Any task can be used to create, sign, and verify archive libraries and integrate them into a CI/CD pipeline.

From the ABL, you can use the `Progress.Archive.ArchiveInfo` class to access the information from a manifest file for a given archive library.

Properties in the Progress.Archive.ArchiveInfo class can be used to access the main attributes of the manifest file: `SignaturePolicy`, `ValidationPolicy`, `Vendor`, `Version`, and others. The `GetValue() method` of the class can be used to access custom attributes.

For a PAS for OpenEdge environment, you can specify archive libraries (`.apl`) in the PROPATH configuration in the `openedge.properties` file. Many libraries shipped with OpenEdge are provided as signed archives in addition to the existing procedure libraries.

Archive libraries can be used to enhance the organization, performance, and security of ABL applications.

Notes:

- Compression is not available for archive files (`.apl`).
- Memory-mapped libraries are only available for procedure libraries (`.pl`) files.

Related Resources:

- Manage Libraries
- Use the PROPACK utility
- Use the PROSIGN utility
- Use Ant tasks to create, sign, and verify archives
- Libraries and PROPATH
- Access information in a manifest file

Using Automatic Database Reconnection with PAS for OpenEdge

The Automatic Database Reconnection functionality was introduced in OpenEdge 12 to support Continuous Operations. This functionality allows for automatically connecting to an alternate database if the connection to the primary database fails. An alternate database is typically a replication target database, but it can also be a backup, or some other database.

This functionality is provided using the `-autoReconnect` parameter. It applies to OpenEdge clients (GUI and TTY). It does not apply to PAS for OpenEdge sessions. However, you can use the Activate Procedure to achieve this behavior with a PAS for OpenEdge session.

The following configuration can used to automatically reconnect to a database in a PAS for OpenEdge application:

1. **Write a parameter file with the configuration to connect to the databases (primary and alternate databases).**

   Example:

   ```
   # autoreconnect.pf -db sports2020 -H dbserver0 -S 20000
   -ct 1 -dbalt1 "sports2020 -H dbserver1 -S 20001" -dbalt2
   "sports2020 -H dbserver2 -S 20002" -retryConnect 1
   -retryConnectPause
   ```

2. **Specify the parameter file in the `openedge.properties` file for the PAS for OpenEdge instance to connect on startup.**

   Example:

   ```
   agentStartupParam=-T "${catalina.base}/temp" -pf /psc/wrk/
   autoreconnect.pf
   ```

3. **Write a procedure file to connect to the database if it is not connected. This program should be available to PAS for OpenEdge through the PROPATH.**

   Example:

   ```
   // dbconnect.p IF NOT CONNECTED("sports2020") THEN CONNECT
   "-pf /psc/wrk/autoreconnect.pf".
   ```

4. **Specify the `dbconnect.p procedure` file in `sessionActivateProc` property in the `openedge.properties` file for the PAS for OpenEdge instance.**

   Example:

   ```
   sessionActivateProc=dbconnect.p
   ```

With the configuration specified above, PAS for OpenEdge can reconnect to the database when a request is performed. Using Automatic Database Reconnection in an ABL application deployed using PAS for OpenEdge can improve the high availability of the application.

Related Resources:

- Learn about automatic database reconnect
- Automatic Database Reconnect for ABL Clients
- Activate procedure

Deploying PAS for OpenEdge Applications to the Cloud

You can deploy PAS for OpenEdge n-tier applications (Database, Data Services, Web UI) to the cloud to increase availability, flexibility, and scalability. There are three migration approaches that are commonly used to deploy applications to the cloud: rehosting, re-platforming, and re-architecting. You can apply these to deploy PAS for OpenEdge applications to the cloud.

With the rehosting approach (also called lift and shift), you would deploy the application to the cloud without modifying it. This is the simplest approach. You would use EC2 instances (AWS) or virtual machines (Azure) to run the servers for your PAS for OpenEdge n-tier application. In this approach, you can use a private cloud (VPC) to have a private network and increase security. You can also use a load balancer to access the application.

With the re-platforming approach, you would change the way that select components of your application are deployed and take advantage of cloud functionality. This requires medium effort. For example, you can make a template for the PAS for OpenEdge servers to use auto-scaling to be able to scale on-demand.

The re-architecting approach corresponds to reviewing the architecture of your application and changing its components to use cloud native functionality. This approach is more involved and requires a greater effort.

You can also use container technologies to deploy PAS for OpenEdge and use container services or managed Kubernetes services. For example, you could build a container image for your application using your CI/CD pipeline (on-premises or on the cloud) and deploy the single image to the cloud using a rolling update as the deployment strategy.

You can use OpenEdge Replication to provide redundancy for the database and deploy the servers on multiple availability zones. The servers for PAS for OpenEdge and Web UI running with auto-scaling functionality can also be deployed on multiple available zones.

Overall, deploying your PAS for OpenEdge application to the cloud can bring benefits to the availability, flexibility and scalability of the application depending.

Related Resources:

- Moving Your OpenEdge Application to the Cloud

Deploying PAS for OpenEdge Applications with OpenEdge Command Center

OpenEdge Command Center (OECC) is a modern web-based console to manage OpenEdge databases and PAS for OpenEdge instances. It is comprised of two major components, the OECC Server and the OECC Agent. You can use it to manage both on-premise and Cloud resources. The OECC Agent also provides support for OpenTelemetry metrics that you can access from popular Application Performance Management tools (APM).

The cloning functionality can be used to clone a Progress Application Server (PAS) for OE instance to multiple OpenEdge installations. This makes it easier to work with load balancing and provides high availability for PAS for OpenEdge.

You can use OECC to deploy and visualize ABL applications (and Web applications) on a PAS for OpenEdge instance.

The OpenTelemetry support of the OECC Agent can be used to query performance metrics for the OpenEdge database and for PAS for OpenEdge from an Application Performance Monitoring (APM) solution such as Elastic APM, Dynatrace, NewRelic and others to view the performance metrics. You can also configure it with Prometheus and Grafana.

OECC also includes a REST API that you can use to automate operations. OpenEdge Command Center simplifies the management of the OpenEdge platform and brings productivity gains for its day-to-day management.

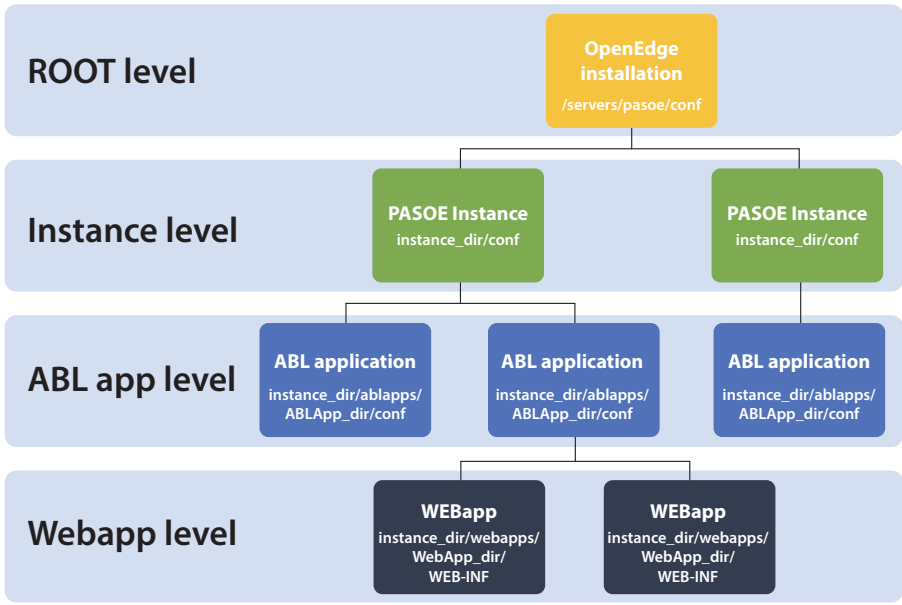
Related Resources:

- Learn about OpenEdge Command Center
- Clone a PAS for OpenEdge instance
- Deploy ABL applications or ABL web applications
- Monitor OpenEdge resources using the OpenEdge Command Center agent

PAS for OpenEdge Configuration

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/pasoe` are copied to the instance home directory `instance-dir`. The instance can then be customized for its ABL application, ABL web applications, ABL service 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 Property files and details on the PAS for OpenEdge Instance structure at Instance directories.

There are several tools available 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:

`openedge.properties`	Application specific properties similar to `ubroker.properties`, and some PAS-specific properties. Edits are always needed here and can be done using OEPROP or OpenEdge Management.
`catalina.properties`	Tomcat specific properties used by the webserver. Small edits such as port values are normally needed here using PASMAN for system-wide commands or TCMAN if you are in the instance directory. See `instance-dir/conf/catalina.properties.README` for more information.
`jvm.properties`	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.
`appserver.properties`	Specifies Java properties used by PAS for OpenEdge and ABL web applications. Edits are not normally needed here but if done use tcman. See `instance-dir/conf/appserver.propertes.README` for more information.

You can find details on configuring your instance at Migrate classic AppServer properties and the property migration utility at Migrate classic AppServer properties to PAS for OpenEdge using PASPROPCONV.

Note: With PAS for OpenEdge, multiple sessions are possible within a single MS Agent. This is in contrast to the one-to-one mapping used by classic AppServer. 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 the environment variables.

PAS for OpenEdge Runtime

PAS for OpenEdge Transports

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

Transport	URL Path	Notes
APSV	**`/apsv`**	Supports the OpenEdge AppServer protocol (binary) over HTTP/S
REST	**`/rest`**	Supports REST RPC using a .paar file; replaces REST Adapter
SOAP	**`/soap`**	Supports SOAP 1.1; replaces WSA
WEB	**`/web`**	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 Transports and Services and related configuration at Deployment artifacts.

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 an 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 is used to control 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: • 405/Method Not Allowed • or 501/Not Implemented	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 in this document. You can also deploy the management web applications, `manager.war` and `oemanager.war`, during the creation of development instances using the `-f` option.

Programming Considerations for PAS for OpenEdge Application Models

The OpenEdge application classic AppServer supported four different server-state caching models in its ABL Sessions: state-reset, state-aware, stateless, and state-free. Each of these offers the OpenEdge client's developer a choice of who maintains state across sequential client requests, with the choices being the ABL Session or the ABL business application. The new ABL Session caching models in PAS for OpenEdge are grouped into two categories of TCP connection managements: Session-managed and Session-free. Session-managed uses a single TCP connection to pipeline execution of all ABL client requests

over that logical connection. Session-free uses a pool of single TCP connections to create the appearance of concurrently executing ABL client requests over that logical connection. It is important to understand the PAS for OpenEdge behavior to properly emulate your existing functionality in the two Application Models available in PAS for OpenEdge.

Note: This facet of application design does not indicate which software architectural model the business application uses because all are stateful architectures, regardless of the name assigned by OpenEdge.

The Session-managed Application model (APSV/SOAP) is designed for business applications to support a single transaction across multiple requests from the same client, returning intermediate results with each request until the transaction is completed. In this case, the client holds a persistent connection to the application server until the transaction is complete. Requests can be routed to multiple ABL sessions unless the connection is bound which will send all requests from the client to the specific ABL session that performed the connection. There are also considerations regarding the request context. Use the SERVER-CONNECTION-CONTEXT, SERVER-CONNECTION-ID, SERVER-CONNECTION-BOUND, and SERVER-CONNECTION-BOUND-REQUEST, attributes to manage context for a Session-managed Application model. You can find more details on session-managed programming at Session-managed programming.

The Session-free Application model (REST/WEB/SOAP) is designed for business applications that return a complete result, starting and ending any transactions, in a single request. Thus, the server maintains no context for any client that it services. Requests from a session-free client are handled by any available ABL session in any available multi-session agent that supports the required business application. The session-free client has each of its server requests executed by an available ABL session chosen by the PAS for OpenEdge session manager from its pool of ABL sessions. In this application model, multiple requests from a single client can be executed in parallel, as PAS for OE resources permit. The more ABL sessions that are available to handle requests for a given business application, the more requests that the application can handle simultaneously from a single client.

If context is required between requests, you can use the Activate and Deactivate event procedures to initialize and clean-up for each request. For context such as a login token or client context, the information must be passed on every request and response to provide the context between ABL sessions since there is no guarantee a request will return to the same ABL session it ran a previous request on. Use the CURRENT-REQUEST-INFO attribute and CURRENT-RESPONSE-INFO attribute to maintain this context for a Session-free Application model. This model supports REST, WEB, and SOAP transports. You can find more details on session-free programming at Session-free programming.

You can find more details on PAS for OpenEdge programming at Programming for a PAS for OpenEdge application model.

Connection Models

Software applications architectures operate using either a stateful or stateless connection model. Stateful models employ a logical/physical connection between a client and server with the expectation that a sequence of client requests maintain state information subsequent requests can use. Stateless models do not employ state information, and each request is a completely self-contained operational unit.

A PAS for OpenEdge client may be bound or un-bound to an ABL Application's single ABL Session. This replicates the Classic AppServer's programming model for stateful client connections. The difference lies in all ABL Sessions operating in Classic AppServer are required to operate in one of the server-state caching models. PAS for OpenEdge lifts that restriction and permits each ABL Application's ABL Session to transition to any of the four server-state caching models per execution of a client request.

The use of HTTP Cookies to synchronize client and server state is a common way stateful web application use to execute sequences of client requests where each request depends on the final state of the previous request. A common web application behavior is to eliminate the Cookies if they have not been used in a certain period, the purpose being to recover server memory and caching resources that can be used for other active clients. The effect this behavior has on the client is that should the HTTP Cookie, or the HTTP Session cached data it references, expire or is deleted, the synchronization of a stateful application's client and server is lost. The net effect is the same as losing a TCP connection – you have to restart the client's logical connection to the server with the knowledge that all state information is lost, and new state information must be initialized. The connection mechanism for both Session-managed and Session-free Application models is the same. You use the CONNECT() method on a server object handle to create the appropriate connection from a client to PAS for OpenEdge. It is important to understand connection modes and how they affect the binding between a client and a PAS for OpenEdge instance.

In classic AppServer, the operating modes supported for client connections are defined in `ubroker.properties` with each classic AppServer instance supporting only one operating mode.

With PAS for OpenEdge, the client controls the connection mode by specifying either the session-managed or session-free mode in the `CONNECT()` method. This means that 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.

When establishing a connection, the client defines the connection mode using a new connection parameter `session-model`. Once connected, requests from that client 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 will create a session-managed connection from the client to PAS for OpenEdge, which automatically runs the specified `CONNECT` and `DISCONNECT` event procedures.

To emulate the state-aware and state-reset behavior, you need to also 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 migrating Operating Modes to Application Models at Migrate classic AppServer operating modes and on binding the connection at Migrate classic state-aware operating mode.

Caching Options

An application can choose whether state information (context) is cached by the client or the server. Client-side state caching is an architectural choice for stateful web applications where the application server generates a block of application state and delivers it to its client, which caches it and passes it to the next application server to execute its next client request. The application server uses the client supplied application state information to execute the request and then updates and returns it to the client so that it can be used by the next client request execution.

Server-side state caching is an architectural choice for stateful web applications where the application server generates a block of application state, and caches it for use to execute subsequent client requests. To synchronize the cached state for exactly one client and its ABL application the server's application will employ the use of HTTP Cookie functionality made available to it by PAS for OpenEdge. The application client's responsibility is to echo the HTTP Cookie it receives along with each application server request and to remove the Cookie when instructed to do so by the application (via the application server). The physical form of cache storage chosen by the application server varies by use-case, but the common design is to pass a reference to its storage, known only to the application server, inside the HTTP Cookie. The variation on this theme is found when the application state is very small in size and can be effectively become the content of the HTTP Cookie.

Connection State and Behavior

The behavior of PAS for OpenEdge is controlled by the client. It depends on the combinations of application architecture, client connection types, client/server-side application state caching, and client-ABL Session binding.

The OpenEdge application classic AppServer supported four different server-state caching models in its ABL Sessions: state-reset, state-aware, stateless, and state-free. Each of these offers the OpenEdge client's developer a choice of who maintains state across sequential client requests, with the choices being the ABL Session or the ABL business application. The ABL Session caching models are grouped into two categories of TCP connection managements: Session-managed and Session-free. Session-managed uses a single TCP connection to pipeline execution of all ABL client requests over that logical connection. Session-free uses a pool of single TCP connections to create the appearance of concurrently executing ABL client requests over that logical connection.

The following table illustrates the combinations of client and server connections and the resulting behavior.

Client			**HTTP/S**	**Server**			
Type	**ABL Session connection model**[1]	**App Stat model**[2]	**State/ID**	**ABL Webapp Transport**	**MS-Agent ABL Session connection type**	**Caching for Stateful clients**	**ABL Session Bound**[3]
OpenEdge: AVM Java, .NET	Session-Managed: State-reset	Stateful		APSV over HTTP	State-reset (manual)	ABL Session	Yes
	Session-Managed: State-aware	Stateful	**State: HTTP Cookie**[4]	APSV over HTTP	State-aware (manual)	ABL Session	Yes
	Session-Managed: Stateless	Stateful	**Identity: HTTP header**	APSV over HTTP	Stateless	Application control	Application control
	Session-Managed: State-free	Stateful		APSV over HTTP	State-free	Application control	Application control
SOAP over HTTP	Session-Managed: State-reset	Stateful	**State: HTTP Cookie**[4] **Identity: HTTP header**	SOAP over HTTP	State-reset	ABL Session	Yes
	Session-Managed: State-aware	Stateful		SOAP over HTTP	State-aware	ABL Session	Yes
	Session-Managed: Stateless	Stateful		SOAP over HTTP	Stateless	Application control	Application control
	Session-Managed: State-free	Stateful	**Identity: HTTP header**	SOAP over HTTP	State-free	Application control	Application control

Client			HTTP/S	Server			
Type	ABL Session connection model	App Stat model	State/ID	ABL Webapp Transport	MS-Agent ABL Session connection type	Caching for Stateful clients	ABL Session Bound
Any program with HTTP support	N/A	Stateless	**State: HTTP Cookie** **Identity: HTTP header**	Rest over HTTP	State-free	Application client-side caching[5]	N/A
					State-free	Application server-side caching[6]	N/A
Any program with HTTP support	N/A	Stateless	**State: HTTP Cookie**[4] **Identity: HTTP header**	Web over HTTP	State-free	Application client-side caching	N/A
					State-free	Application server-side caching	N/A

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 TCMAN Reference.

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.

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.

HealthScanner

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

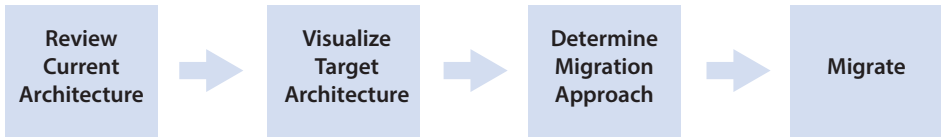
Deferred logging is a feature of PAS for OpenEdge to record stack trace information immediately preceding a multi-session agent crash. It also can be used to run an on-demand monitoring check on an instance through an API. Deferred logging keeps a separate memory buffer from the regular agent logging. 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.

Application Monitoring

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 Monitor PAS for OpenEdge instances.

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 recommended 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 on the same business logic and databases

Other important questions to consider:

- Architecture

  Does your application need 32-bit Windows support?

  Are there 3rd party components that need special consideration?

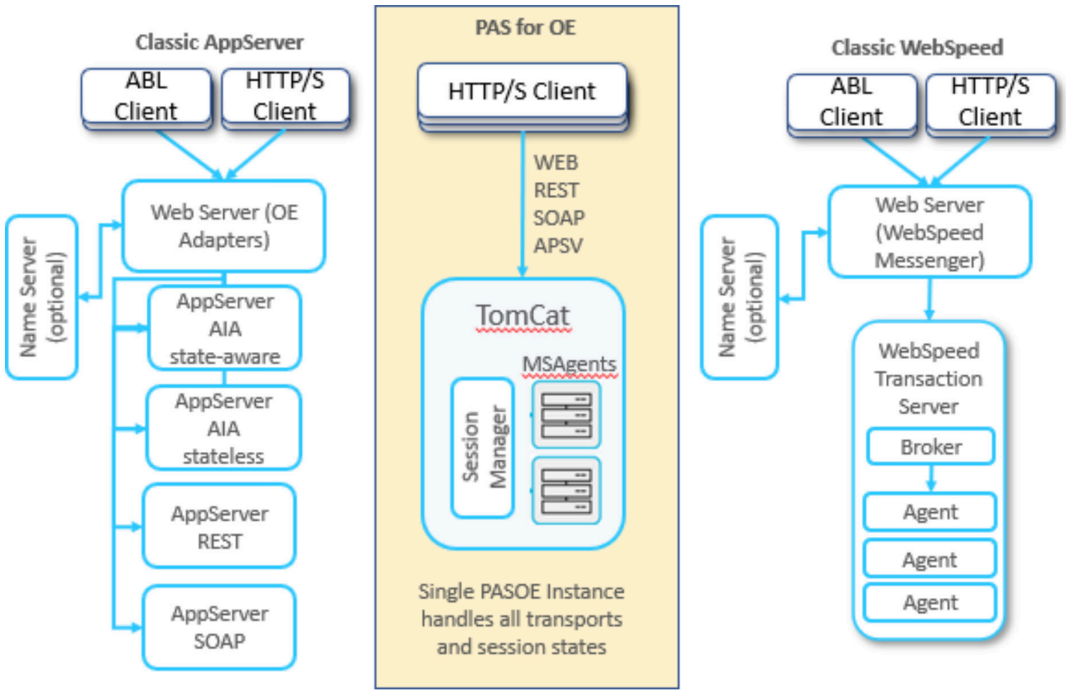
- Deployment
  - Runs on-premise, in the cloud, or hybrid?
  - How many servers are needed to run the application?
  - Are you running on Linux, Windows, AIX, and Solaris?
  - TCP Ports, Firewall Rules.
  - How is the application built and packaged?
  - Are you using a CI/CD pipeline for deployment?

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, Load Balancing, etc.
- Firewall rules for new ports (classic AppServer versus 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

When replacing classic AppServer and WebSpeed components with PAS for OpenEdge, bear in mind that a single PAS for OpenEdge ABL Application can replace multiple classic AppServers and WebSpeed, since PAS for OpenEdge does not define 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. Progress Professional Services can assist you in this journey following a prescriptive modernization approach that follows a suggested reference architecture. They can help identify 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

It is important to plan your migration and become familiar with the migration steps and differences between your current and target architectures before beginning your migration. You can find many migration documents at KB Article: Migrating to OpenEdge 12.

Move to OpenEdge 12.8 and PAS for OpenEdge

You can do a migration directly to OpenEdge 12.8 from 11.7 classic AppServer and WebSpeed applications to PAS for OpenEdge including your database and environment. During migration you can 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 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 and later)
- 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 vastly different runtime environment than classic AppServer and WebSpeed. It is often 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 opportunity 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 you will test this. Also consider this opportunity 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 useful tips and additional material references at KB Article: 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:

Name	Purpose
`paspropconv`	Harvest/convert properties from classic AS/WS (1-Time)
`tcman`	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`	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 versus 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 becomes 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 run-time 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 KB: Performance issues between PAS 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 relies on load-balancing multiple, smaller servers to satisfy increased demand only when the situation requires. The answer to this problem is scaling, but first we still need to establish a suitable machine for 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 Tune PAS for OpenEdge instances. Additional details on Server Sizing can be found later in this document at **Server Sizing**.

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, there 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) versus 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. The use of a load-balancer can also be considered to improve throughput.

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.

Once it is understood how an application should perform at peak with specific resources, it should be possible to extrapolate a scaled environment.

ABL Client Migration

In the classic OpenEdge AppServer, the operating modes for session-managed client connections are 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 a 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

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 at Develop an ABL service using the WEB transport.

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.

Using WEB Transport for 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 of existing REST services or create new REST Services using the Web transport as detailed in Develop an ABL service using the WEB transport.

SOAP Client Migration

You can migrate existing SOAP ABL Services to the PAS for OpenEdge SOAP transport. ABL SOAP clients for PAS for OpenEdge connect using the state-managed application model. 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://localhost:8810/wsa/wsa1/wsdl ?targetURI=urn:CustSvc`

ROOT Web App: `-WSDL https:// localhost:8810/soap/ wsdl?targetURI=urn:CustSvc`

Named Web App: `-WSDL https:// localhost:8810/CustMaint /soap/wsdl ?targetURI=urn:CustSvc`

You can find details on SOAP client connections at Migrate WSA URLs to use the SOAP transport. 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

There are some significant differences between classic WebSpeed and PAS for OpenEdge. Primarily, 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.

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.

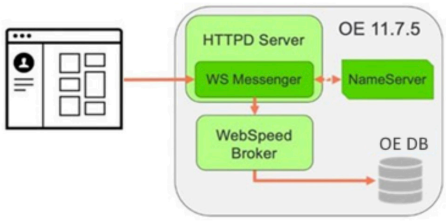
You can find details at Migrate Classic WebSpeed Applications, What's new and different in WebSpeed on PAS for OpenEdge and Differences between classic AppServer, WebSpeed Transaction Server and PAS for OpenEdge.

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 require a conversion of the code base.

Architecture Comparison

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

Classic WebSpeed

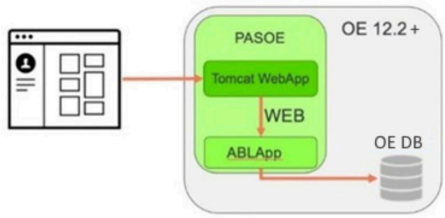


Requests are initially serviced by a web server for the WebSpeed Messenger.

Optionally, a NameServer may be consulted to determine the appropriate WebSpeed broker.

Requests are routed to the WebSpeed broker to execute business logic against a database.

PAS for OpenEdge



Requests are always directed to a Web Application within the PAS for OpenEdge instance.

WebSpeed requests are handled using the WEB transport along with a WebHandler.

The business logic executes against a connected database.

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 available for use:

- 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
```

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.

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 for 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`

Migration of WebSpeed components

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 on OpenEdge
Static Files (HTTPD Server)	*instance-dir*/webapps/webapp-name/static/*folder*
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)
ubroker.properties	openedge.properties (paspropconv)
web-disp.p	web-handler.p OpenEdge.Web.CompatibilityHandler or Custom WebHandler

High-level Migration Steps for WebSpeed Applications

To migrate a classic WebSpeed application to PAS for OpenEdge you can follow the steps defined in Migrate Classic WebSpeed Applications. In general, you will be performing the following tasks:

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.

   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, to map the old URL to the new ones which are prescriptive. 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" />
   ```

1. 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 10 at Apache Tomcat 10: The rewrite value.

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 information that explains these parameters and their settings in `$CATALINA _ BASE/conf/openedge.properties.README`.

In the WEB transport, requests are dispatched to the ABL procedure 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 and later, WebHandlers are configured with the ABL web application deployed to an instance. Use this properties file `instance-dir/webapps/webapp-name/WEB-INF/adapters/web/service-name/service-name.handlers` to map URLs to specific WebHandlers. 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`.

Mapping of Classic WebSpeed URIs to PAS for OE URIs

In the Classic WebSpeed environment a web server would utilize the WebSpeed Messenger and a service script to direct requests for an application to the correct WebSpeed broker. This occurs by using a CGI script which defines the WebSpeed service to be used, and some application code may use the SelfURL variable to refer to the URI of the current page. However, PAS for OpenEdge follows a more traditional URI pattern using webapp names as part of the path, and reserved path names which represent transport adapters, so the URI patterns are unavoidably incompatible.

In the situation where users or other applications may need to rely on these old URI patterns, we need to modify requests on the fly in a way that PAS for OpenEdge can both understand the request while the ABL code can react to the old service URI. To do this, we can override some internal behavior without altering the product code directly. We accomplish this by leveraging some existing behaviors which still exist in WebSpeed for PAS for OpenEdge: utilizing the SUPER_PROC environment variable to adjust the super procedures loaded by WebSpeed and rewriting the requested URIs from the user via the web server.

First, we need to define the `SUPER _ PROC` environment variable which will add a procedure to the persistent procedure stack. To add the environment variable, you must leverage the startup process for the Tomcat server which looks for files in the `CATALINA _ BASE/bin directory` which contain the `suffix _ setenv.[bat|sh]`. Modifying existing scripts is not recommended, so a new file should be created as `CATALINA _ BASE/bin/webspeed _ setenv.[bat|sh]` and will define an OS-appropriate environment variable. In the examples below, this environment variables point to a new procedure file called `selfurl.p` which we will create in a later step.

- For Windows, create a `webspeed _ setenv.bat` script file in the `CATALINA _ BASE/bin` directory to set the relevant environment variable to the new procedure with the following line:

```
set SUPER _ PROC=slurp
```

- For Unix/Linux, create a `webspeed _ setenv.sh` script file in the `CATALINA _ BASE/bin` directory to set the relevant environment variable to the new procedure with the following line:

```
export SUPER _ PROC=slurp
```

Next, we need to create the procedure file slurp which will be used to override and extend the indicia procedure. This internal procedure will convert common CGI variables into ABL global variables that are accessible from anywhere in the application code. The override procedure will first execute the original code (via RUN SUPER) then always attempt to extract a query parameter called cgi-script from the requested URI if it is present. Place this file into the PROPATH for the PAS for OpenEdge instance:

```
// File: selfurl.p

BLOCK-LEVEL ON ERROR UNDO, THROW.

{src/web/method/cgidefs.i} // Basic CGI variables

PROCEDURE init-cgi:
  RUN SUPER.

  // example is a request for GET /cgi-bin/svcname/sample.p
  // the rewrite rules changes that to
  //   GET /web/sample.p?cgi-script=svcname

  IF INDEX(WEB-CONTEXT:GET-CGI-LIST('QUERY':u), 'cgi-script') GT 0 THEN
  DO:
    // change the SCRIPT_NAME from /web back to /cgi-bin/svcname
    SCRIPT_NAME = SUBSTRING(SCRIPT_NAME, 1,
                            R-INDEX(SCRIPT_NAME, '/'))
                + 'cgi-bin/'
                + WEB-CONTEXT:GET-CGI-VALUE('QUERY':u,
                                            'cgi-script').

    // PATH_INFO is /sample.p (after /web) so add that on
    SelfURL = SCRIPT_NAME + PATH_INFO.
  END.

  MESSAGE 'After/SCRIPT_NAME:' SCRIPT_NAME.
  MESSAGE 'After/SelfURL:' SelfURL.

END PROCEDURE.
```

The goal of this override is to extract the service name from a query parameter cgi-script as the global variable `SCRIPT _ NAME` value. This can then be combined into the standard global variable `SelfURL` and can be used within the ABL application.

Lastly, we need to provide the URI rewriting rules to convert from Classic WebSpeed to the new PAS for OpenEdge URI pattern. Place the following into a new file `rewrite.config` int the `WEB-INF/` directory of the webapp where the WebSpeed application is served.

```
RewriteRule (.)/cgi-bin/(.)/(.*) $1/web/$3?cgi-script=$2 [QSA]
```

This replaces the Classic WebSpeed URI of `/cgi-bin/script/path/to/program.p`

with `/web/path/to/program.p?cgi-script=script`

With these changes in place, it should be possible to allow existing links to your application to continue working with the new PAS for OpenEdge instance, until such time as you can have users or applications update to the new URI pattern.

Open Client Migration

PAS for OpenEdge supports session-managed and session-free application models which provide OpenClient with 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://localhost:8810/`**`apsv`**
Named Web App: `https://localhost:8810/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 and later, 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 OpenEdge 12 the exception handling has been enhanced to return the error that was raised on the AppServer. Open Client distinguishes between application, STOP, and system errors. These can be identified separately, and include the exact error message from the error raised by the ABL. If you have implemented the previous exception handling, that will still work. No coding changes are required to take advantage of the new exception handling.

- **`Object.finalize()`** method has been removed from some classes as the java method was deprecated and the use of finalize is no longer considered good practice. In some cases, 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 `Open4GLException`** and **`Open4GLError`** (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 and `AutoClosable`. Use the OpenEdge 12.2 or later ProxyGen with your existing proxy projects [`.xpxg`] to generate new executables. You may continue to use older generated proxies with the updated 12 Open Client implementation, but it will not take advantage of the updates.

- **Performance Improvements** in OpenEdge version 12.5 for Java Open Client have reduced garbage collection and improved concurrency by reducing locking. Both String concatenation and Integer creation, especially for logging, have been greatly reduced. This improves the CPU and memory overhead required for GC for java applications that use Open Client. Locking in high contention areas has been reduced, improving multiple request concurrency, such as when Java Open Client is deployed inside a web server.

- **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.

- **Javadoc** for Java Open Client has been published on the OpenEdge documentation website.

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 a supported version of Java. Once you have generated the class files, you can now import them into your project. Since the method signatures have not changed you can 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 or HTTPS. You will have to change all 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 Connection objects 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.

Step 4 – Deploy and test applications.
At this point you should be able to deploy your application to a test environment and run tests to verify your code is functioning correctly.

Server Sizing

Configuration

Performance 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 versus 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 versus 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
`maxAgents=2`	Maximum number of multi-session agents to start Instances
`minAgents=2`	Start a new agent if availability is below this value
`numInitialAgents=2`	Start 2 initially to satisfy the min/max configured Initial number of multi-session agents to start
`agentStartLimit=1`	Starts only 1 agent at a time if below minAgents
`maxABLSessionsPerAgent=20`	Default is 200 in OpenEdge 12.2 and later. Maximum ABL sessions per multi- session agent
`maxConnectionsPerAgent=20`	
`numInitialSessions=4 – Start max 20% of maxConnectionsPerAgent`	Default is 200 in OpenEdge 12.2 and later Maximum connections per multi-session agent
`minAvailableABLSessions=3`	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 on 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 details on PAS for OpenEdge tuning at Tune PAS for OpenEdge instances.

Performance Testing

Proper Sizing Based on Performance Tests

- Performance Testing – Ensure code is responding in a timely manner
  - Memory leaks (e.g., 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 KB: PAS for OpenEdge - Example of PAS for OpenEdge Machine sizes for ATM Benchmark on Amazon AWS) 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, adjusted 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. In the KB: PAS for OpenEdge - Sizing Your Machine (ATM Benchmark) 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 Tune PAS for OpenEdge instances 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 REST API Reference for oemanager.war – 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.
- OpenEdge Memory Profiler – Can be used to analyze the memory usage of ABL applications and identify excessive memory consumption and memory leaks.

Troubleshooting

Using the Progress OpenEdge Memory Profiler

The OpenEdge Memory Profiler is available starting with OpenEdge 12.8.

You can use this new functionality to analyze the memory usage of ABL applications and identify excessive memory consumption and memory leaks.

OpenEdge clients (GUI and TTY) and PAS for OpenEdge can generate Memory Profiler files (.oemp extension). These files contain data (snapshots) on the memory usage of the ABL applications. The files can be imported into the OpenEdge Memory Profiler Tooling (OEMP) to visualize them. OEMP is a new web-based application with a PAS for OpenEdge-based backend. The Memory Profiler files and their representation in OEMP are known as recordings.

Steps to use the OpenEdge Memory Profiler:

1. Generate a Memory Profiler file.
2. Import the Memory Profiler file into the OpenEdge Memory Profiler Tooling.
3. Visualize the recording.

You can find details at Profile memory.

Generating a Memory Profiler File

You can generate Memory Profiler files using the new parameter `-profileMemory` with an OpenEdge client (GUI and TTY) and PAS for OpenEdge. You can run an ABL application either interactively, batch, or server-based (PAS for OpenEdge). Example:

```
mpro sports2020 -p program.p -profileMemory config applicationURL=/cgi-
bin
ls -l memprof.*.oemp
```

For PAS for OpenEdge, you can specify the `-profileMemory` as part of the `agentStartupParam` property. Example:

```
agentStartupParam=-T "${catalina.base}/temp" -db sports2020 -S 20000
-profileMemory /psc/wrk/config
```

Notes:

- The `-profileMemory` parameter can only be specified as a startup parameter and not in a `.pf` file.
- The "config" file can be an empty file. In this mode a default configuration is used.
- By default, the `.oemp` file is stored in the current working directory for ABL clients and stored in the work folder for a PAS for OpenEdge environment.
- A Memory Profiler file has the format `memprof.<pid>.oemp` for ABL clients and `memprof.<pid>.AS-<agentid>.oemp` for a PAS for OpenEdge environment.

Importing the Memory Profiler file into the OpenEdge Memory Profiler Tooling

You can copy the generated .oemp file to the OEMP backend upload or import folder(s) and use the Web UI to import them to make them ready for visualization.

The OEMP backend can be configured to automatically import the files.

Visualizing the recording

Using the Web UI, you can show a chart for the snapshots in a recording using the OpenEdge Memory Profiler Tooling, view snapshot details and compare snapshots.

You can use the Memory Profiler in your applications running in a CI/CD pipeline to monitor the memory utilization automatically and identify issues between builds of the application.

Next Steps

The information in this guide is meant to start the 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 **Progress 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 WebHandler on PAS for OpenEdge
- Progress Application Server for OpenEdge Developers
- Progress Application Server for OpenEdge Administration (PAS for OpenEdge)

About Progress

Progress (Nasdaq: PRGS) empowers organizations to achieve transformational success in the face of disruptive change. Our software enables our customers to develop, deploy and manage responsible AI-powered applications and digital experiences with agility and ease. Customers get a trusted provider in Progress, with the products, expertise and vision they need to succeed. Over 4 million developers and technologists at hundreds of thousands of enterprises depend on Progress. Learn more at www.progress.com

/progresssw
/progresssw
/progresssw
/progress-software
/progress_sw_

Progress