Progress®



OpenEdge® Service Pack 11.7.4:
New Information

OpenEdge®

Copyright

© 2018 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

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

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

Please refer to the Release Notes applicable to the particular Progress product release for any third-party acknowledgements required to be provided in the documentation associated with the Progress product.

The Release Notes can be found in the OpenEdge installation directory and online at: https://community.progress.com/technicalusers/w/openedgegeneral/1329.openedge-product-documentation-overview.aspx.

For the latest documentation updates see OpenEdge Product Documentation on Progress Communities: (https://community.progress.com/technicalusers/w/openedgegeneral/1329.openedge-product-documentation-overview.aspx).



October 2018

Last updated with new content: Release 11.7.4

Updated: 2021/06/28

Table of Contents

Preface....7
Purpose....7
Audience....8
Organization....8
Use ABL documentation....8
References to ABL compiler and run-time features....8
References to ABL data types....9
Typographical conventions....9
Examples of syntax descriptions....10
Long syntax descriptions split across lines....12
Complex syntax descriptions with both required and optional elements....12
Example procedures....12
OpenEdge messages....13
Obtain more information about OpenEdge messages....14

What's New in 11.7.4....15
Startup Parameters....15
Join Hint (-nojoinhint)....16
OpenEdge ABL....16
Support for HMAC SHA algorithms in MESSAGE-DIGEST function....16
OpenEdge Data Object Handler....16
Introduction....17
Create an ABL service....18
Create a mapping file....19
Define custom service endpoints....19
Associate HTTP verbs with operation handlers....20
How to use operation handlers....22
OpenEdge DataServers....30
Performance Improvement for inner join query having FIRST/LAST construct at any join level.31
-nojoinhint startup parameter....31
OpenEdge Management....31
Create a PAS for OpenEdge instance....31
Manage PAS for OpenEdge instance....34
Progress Application Server for OpenEdge....41
Improve performance with new tuning and sizing guides....42
Manage and monitor an instance....42
Secure an instance....60
Secure online deployment of a new ABL application 73

Troubleshoot problems....74
Docker Container....91

Preface

For details, see the following topics:

- Purpose
- Audience
- Organization
- Use ABL documentation
- Typographical conventions
- Examples of syntax descriptions
- Example procedures
- OpenEdge messages

Purpose

OpenEdge® Service Pack 11.7.4: New Information documents important information about the OpenEdge Release 11.7.4 Service Pack, with reference to new and existing documentation for Release 11.7 as required. Release 11.7 documentation set references might include:

- Product manuals
- HTML-based online help
- Web papers

References to these documents attempt to identify sections or topics that you can use in searches of the product documentation set.

Audience

This guide is primarily intended for OpenEdge application developers and system administrators who have installed 11.7.4 as the most recent Service Pack for Release 11.7.

Organization

This book contains the following major sections:

- Startup Parameters

  Contains information about new startup parameters.

- OpenEdge ABL

  Contains information about updates to ABL features.

- OpenEdge Data Object Handler on page 16

  Contains information about the Progress OpenEdge Data Object Handler.

- OpenEdge DataServers

  Contains information about how the DataServer for MS SQL Server handles rollback of sub-transactions.

- OpenEdge Management

  Contains information about how OpenEdge Management provides support for CDC User Identity.

- Progress Application Server for OpenEdge

  Contains information about new features in PAS for OpenEdge.

Use ABL documentation

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is *ABL (Advanced Business Language)*. With few exceptions, all keywords of the language appear in all UPPERCASE, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.

For the latest documentation updates go to https://docs.progress.com.

References to ABL compiler and run-time features

ABL is both a compiled and an interpreted language that executes in a run-time engine. The documentation refers to this run-time engine as the *ABL Virtual Machine (AVM)*. When the documentation refers to ABL source code compilation, it specifies *ABL* or *the compiler* as the actor that manages compile-time features of the language. When the documentation refers to run-time behavior in an executing ABL program, it specifies *the AVM* as the actor that manages the specified run-time behavior in the program.

For example, these sentences refer to the ABL compiler's allowance for parameter passing and the AVM's possible response to that parameter passing at run time: "ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error." The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: "The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets."

References to ABL data types

ABL provides built-in data types, built-in class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all UPPERCASE, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.
- Wherever *integer* appears, this is a reference to the `INTEGER` or `INT64` data type.
- Wherever *character* appears, this is a reference to the `CHARACTER`, `LONGCHAR`, or `CLOB` data type.
- Wherever *decimal* appears, this is a reference to the `DECIMAL` data type.
- Wherever *numeric* appears, this is a reference to the `INTEGER`, `INT64`, or `DECIMAL` data type.

References to built-in class data types appear in mixed case with initial caps, for example, `Progress.Lang.Object`. References to user-defined class data types appear in mixed case, as specified for a given application example.

Typographical conventions

This documentation uses the following typographical and syntax conventions:

Convention	Description
Bold	Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.
Italic	Italic typeface indicates the title of a document, or signifies new terms.
SMALL, BOLD CAPITAL LETTERS	Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, **GET** and **CTRL**.
KEY1+KEY2	A plus sign between key names indicates a **simultaneous** key sequence: you press and hold down the first key while pressing the second key. For example, **CTRL+X**.
KEY1 KEY2	A space between key names indicates a **sequential** key sequence: you press and release the first key, then press another key. For example, **ESCAPE H**.
Syntax:	
`Fixed width`	A fixed-width font is used in syntax, code examples, system output, and file names.

Convention	Description
Fixed-width italics	Fixed-width italics indicate variables in syntax.
Fixed-width bold	Fixed-width bold italic indicates variables in syntax with special emphasis.
UPPERCASE fixed width	ABL keywords in syntax and code examples are almost always shown in upper case. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class.
Period (.) or colon (:)	All statements except DO, FOR, FUNCTION, PROCEDURE, and REPEAT end with a period. DO, FOR, FUNCTION, PROCEDURE, and REPEAT statements can end with either a period or a colon.
[ ]	Large brackets indicate the items within them are optional.
[ ]	Small brackets are part of ABL.
{ }	Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.
{ }	Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.
|	A vertical bar indicates a choice.
...	Ellipses indicate repetition: you can choose one or more of the preceding items.

Examples of syntax descriptions

In this example, ACCUM is a keyword, and *aggregate* and *expression* are variables:

Syntax

```
ACCUM aggregate expression
```

FOR is one of the statements that can end with either a period or a colon, as in this example:

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name.
END.
```

In this example, STREAM *stream*, UNLESS-HIDDEN, and NO-ERROR are optional:

Syntax

```
DISPLAY [ STREAM stream ] [ UNLESS-HIDDEN ] [ NO-ERROR ]
```

In this example, the outer (small) brackets are part of the language, and the inner (large) brackets denote an optional item:

Syntax

```
INITIAL [ constant [ , constant ] ]
```

A called external procedure must use braces when referencing compile-time arguments passed by a calling procedure, as shown in this example:

Syntax

```
{ &argument-name }
```

In this example, EACH, FIRST, and LAST are optional, but you can choose only one of them:

Syntax

```
PRESELECT [ EACH | FIRST | LAST ] record-phrase
```

In this example, you must include two expressions, and optionally you can include more. Multiple expressions are separated by commas:

Syntax

```
MAXIMUM ( expression , expression [ , expression ] ... )
```

In this example, you must specify MESSAGE and at least one *expression* or SKIP [(*n*)], and any number of additional *expression* or SKIP [(*n*)] is allowed:

Syntax

```
MESSAGE { expression | SKIP [ ( n ) ] } ...
```

In this example, you must specify {*include-file*, then optionally any number of *argument* or &*argument-name* = "*argument-value*", and then terminate with }:

Syntax

```
{ include-file
    [ argument | &argument-name = "argument-value" ] ... }
```

Long syntax descriptions split across lines

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

In this example, `WITH` is followed by six optional items:

Syntax

```
WITH [ ACCUM max-length ] [ expression DOWN ]
  [ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]
  [ STREAM-IO ]
```

Complex syntax descriptions with both required and optional elements

Some syntax descriptions are too complex to distinguish required and optional elements by bracketing only the optional elements. For such syntax, the descriptions include both braces (for required elements) and brackets (for optional elements).

In this example, `ASSIGN` requires either one or more *`field`* entries or one *`record`*. Options available with *`field`* or *`record`* are grouped with braces and brackets:

Syntax

```
ASSIGN   { [ FRAME frame ] { field [ = expression ] }
           [ WHEN expression ] } ...
       | { record [ EXCEPT field ... ] }
```

Example procedures

OpenEdge documentation may provide example code that illustrates syntax and concepts. You can access many of the example files, and details for installing them, from the following locations:

- A self-extracting Documentation and Samples file available on the OpenEdge download page of the Progress Software Download Center

- The OpenEdge Product Documentation Overview page on Progress Communities:

https://community.progress.com/technicalusers/w/openedgegeneral/1329.openedge-product-documentation-overview.aspx

Once installed, you can locate the example files in the following paths under the OpenEdge Documentation and Samples installation directory:

This directory . . .	Contains examples for the following documents . . .
`src\prodoc\dotnetobjects`	*OpenEdge Development: GUI for .NET Programming*
`src\prodoc\dynamics`	The Progress Dynamics documentation
`src\prodoc\getstartoop`	*OpenEdge Development: Object-oriented Programming*
`src\prodoc\handbook`	*OpenEdge Getting Started: ABL Essentials*
`src\prodoc\interfaces`	*OpenEdge Development: Programming Interfaces*
`src\prodoc\json`	*OpenEdge Development: Working with JSON*
`src\prodoc\langref`	*OpenEdge Development: ABL Reference*
`src\prodoc\prodatasets`	*OpenEdge Development: ProDataSets*
`src\prodoc\visualdesigner`	*OpenEdge Getting Started: Introducing Progress Developer Studio for OpenEdge Visual Designer*
`src\prodoc\xml`	*OpenEdge Development: Working with XML*
`src\samples\open4gl\java`	*OpenEdge Development: Java Open Client*

OpenEdge messages

OpenEdge displays several types of messages to inform you of routine and unusual occurrences:

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.
- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.
- **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

After displaying a message, OpenEdge proceeds in one of several ways:

- Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.
- Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.

- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

OpenEdge messages end with a message number in parentheses. In this example, the message number is `200`:

```
** Unknown table name table. (200)
```

If you encounter an error that terminates OpenEdge, note the message number before restarting.

Obtain more information about OpenEdge messages

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

- Choose **Help** > **Recent Messages** to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.
- Choose **Help** > **Messages** and then type the message number to display a description of a specific OpenEdge message.
- In the Procedure Editor, press the **HELP** key or **F1**.

On UNIX platforms, use the OpenEdge `pro` command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

To use the pro command to obtain a message description by message number:

1. Start the Procedure Editor:

   ```
   OpenEdge-install-dir/bin/pro
   ```

2. Press **F3** to access the menu bar, then choose **Help** > **Messages**.
3. Type the message number and press **ENTER**. Details about that message number appear.
4. Press **F4** to close the message, press **F3** to access the Procedure Editor menu, and choose **File** > **Exit**.

1

What's New in 11.7.4

This topic describes the new features included in OpenEdge Release 11.7.4.

For details, see the following topics:

- Startup Parameters
- OpenEdge ABL
- OpenEdge Data Object Handler
- OpenEdge DataServers
- OpenEdge Management
- Progress Application Server for OpenEdge

Startup Parameters

The OpenEdge Release 11.7.4 Service Pack includes a new startup parameter.

Join Hint (-nojoinhint)

Use Join Hint (`-nojoinhint`) to override the USE_MERGE default query-tuning hint for OUTER-JOIN queries.

Operating system and syntax	UNIX / Windows	`-nojoinhint`		
Use with	**Maximum value**	**Minimum value**	**Single-user default**	**Multi-user default**
DataServer Client	—	—	—	—

For the DataServer for Oracle only, `-nojoinhint` disables the USE_MERGE default query-tuning hint for OUTER-JOIN queries.

If only a few records are needed out of a large result set, the USE_MERGE default can be inefficient. Disable the USE_MERGE hint by compiling your ABL application with the `-nojoinhint` client startup parameter.

The `-nojoinhint` parameter is for compile time only. It has no impact on run time.

OpenEdge ABL

OpenEdge Release 11.7.4 Service Pack adds support for the ABL feature described in the section that follows.

Support for HMAC SHA algorithms in MESSAGE-DIGEST function

The `MESSAGE-DIGEST` function now supports HMAC (Hash-based Message Authentication Code) algorithms. The additional algorithms include *`HMAC-SHA-1`*, *`HMAC-SHA-256`*, *`HMAC-SHA-384`*, and *`HMAC-SHA-512`*. The size of the resulting message digest depends on the algorithm selected, as described in the following table:

Hash algorithm	Return value
`HMAC-SHA-1`	RAW 20-byte binary message digest
`HMAC-SHA-256`	RAW 32-byte binary message digest
`HMAC-SHA-384`	RAW 48-byte binary message digest
`HMAC-SHA-512`	RAW 64-byte binary message digest

OpenEdge Data Object Handler

This section contains new documentation for the Progress OpenEdge Data Object Handler that is included in the OpenEdge Release 11.7.4 Service Pack.

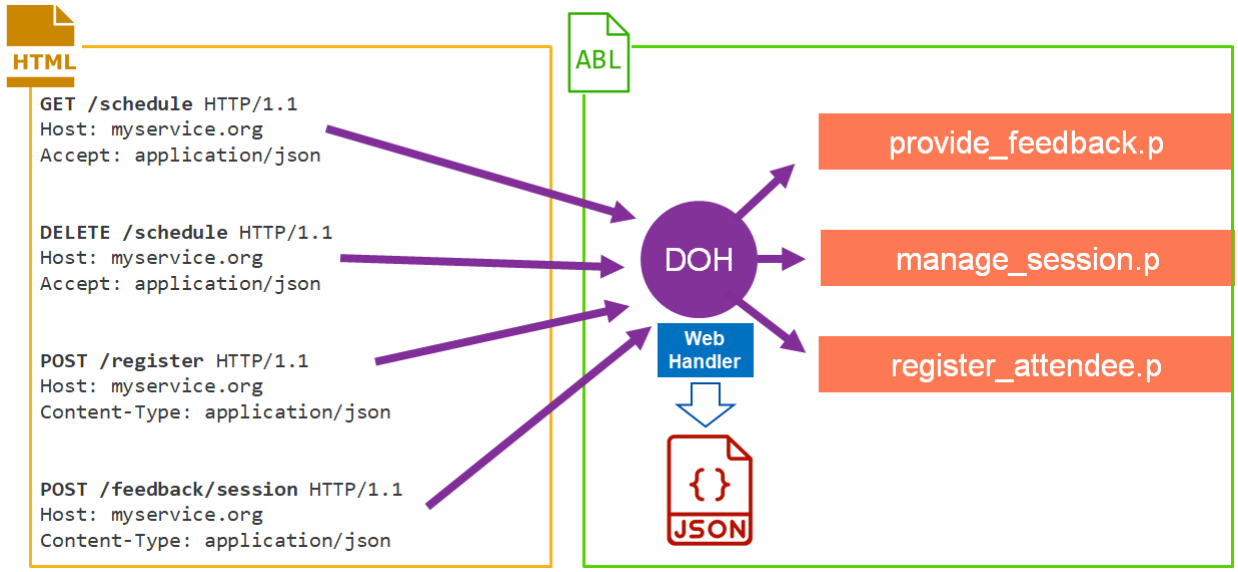
Introduction

The Progress OpenEdge Data Object Handler (DOH) is a built-in Web Handler that enables you to design your own ABL service interface and expose ABL data objects as RESTful resources in a configurable way. It is available via the **WEB** transport in Progress Application Server for OpenEdge (PAS for OpenEdge).

If you have developed Progress ABL Data Object Services, you have already used the DOH under the hood.

How the DOH works

The DOH class implements the `Progress.Web.IWebHandler` interface, and is a more dynamic and business-ready form of the `OpenEdge.Web.WebHandler` class. With the DOH, you create an ABL service, set it to use the `DataObjectHandler` class, and then map custom service endpoints with ABL classes and procedures in a JSON mapping file.



At runtime, when a service endpoint is called, the DOH transforms the HTTP request into an ABL object and invokes the associated ABL class or procedure. It also transforms the output from the ABL class or procedure into an HTTP response.

Overview of steps to use the DOH

1. **Create an ABL service** — You begin by creating an ABL service that uses the DOH class.
2. **Create a mapping file** — You then create a mapping file with the same name as the ABL service.
3. **Define custom service endpoints** — You start describing the service interface in the mapping file by defining your custom endpoints.
4. **Associate HTTP verbs with operation handlers** — For each service endpoint, you associate an HTTP verb (GET, PUT, POST, etc) with one of the following operation handlers:
   - `void`, that returns only a status code
   - `file`, that returns the contents of a file
   - `entity`, that invokes an ABL class or procedure

Should I use DOH or write my own Web Handler?

For an ABL developer, the Data Object Handler relieves the burden of transforming HTTP requests and responses into ABL objects. It does so in a standard way. However, if you have very specific needs, writing your own Web Handler may be the right solution.

Should I use the DOH or a mapped REST service?

Although you can also define a custom service interface using a mapped REST service (that uses the PAS for OpenEdge **REST** transport), a DOH-based ABL service offers the following advantages over the mapped REST service:

- It supports more content types.
- It is easier to debug, since the DOH is an ABL class. You can also readily inspect and modify its JSON mapping file.
- It is more customizable; you can extend its functionality by accessing its class events. To learn more about these class events, see the white paper on using the Data Object Handler.
- It uses the WEB transport, which is the recommended way to expose your ABL applications as RESTful services. This is because the WEB transport uses the ABL `WebHandler` class and thus provides better insight and control over your ABL service.

Create an ABL service

Before you create an ABL service, set up a PAS for OpenEdge instance that you can deploy the service to. You also need to create an ABL Web App project in Progress Developer Studio.

PAS for OpenEdge provides two transports to work with HTTP requests--**REST** and **WEB**. An ABL service is a transport-dependent grouping of resources. For example, imagine that you have two ABL classes —`Order` and `Customer`. You can set up an ABL service to provide access to these classes via a specific transport (REST or WEB, but not both).

The REST transport accepts HTTP requests via a `/rest` URI segment (`http://localhost/myWebApp/rest`). The WEB transport does so via a `/web` URI segment (`http://localhost/myWebApp/web`).

A DOH-based ABL service uses the WEB transport. Once the service is up and running, client applications access the service by appending the service name to the WEB transport URI (for example, `http://localhost/myApp/web/`*`HelloService`*).

Typically though, client applications access a service endpoint, which returns some data. In the URI, the service endpoint follows the service name (`http://localhost/myApp/web/`*`HelloService/CustomerMessage`*).

The DOH enables you to define custom service endpoints and map HTTP verbs to operation handlers for each endpoint. For instance, for an endpoint like `/CustomerMessage`, you may want to map its GET verb with an entity operation handler, and configure the entity handler to use a `getPersonalizedMessage()` method from a `CustomerMessage.cls` class. Similarly, you may want to map its PUT verb with an `updateCustomerMessage()` method, its POST verb with an `newCustomerMessage()` method, and so on.

The first step is to create an ABL service that uses the DOH. Perform this task as follows:

1. Right-click the root project name in the **Project Explorer** view in Progress Developer Studio and select **New > ABL Service**.
2. Select **WEB** in the **Transport** dropdown.
3. Enter a name for the service, (such as HelloService).

4. In the **Web Handler** section, choose **Select existing** and then browse and select **OpenEdge.Web.DataObject.DataObjectHandler**.
5. Add a single resource URI that matches the service name (/HelloService).
6. Click **Finish**.

Create a mapping file

After creating a DOH-based ABL service, you describe its service interface in JSON in a mapping file with the extension `.map`. The DOH uses this mapping file to execute the service's operations. If a client application attempts to call an endpoint or service that is not described in the mapping file, the DOH returns a `404` (Not Found) error.

Create a new text file in your ABL web app project's `/PASOEContent/WEB-INF/openedge` folder with the same name as the ABL service and with the extension `.map` (for example, **HelloService.map**). The matching filename instructs the DOH to use the mapping file for the service's operations. Placing it in the "**openedge**" folder makes it discoverable in the application PROPATH.

Initialize the service

At minimum, you need to define a single service within a `services` object. Like this:

```
{    "services":
      {
        "HelloService": {
        "version": "1.0.0"
      }
    }
 }
```

The service name must match the ABL service and resource URI name that you set while creating the ABL service. The "**version**" property defines the version of the service and should follow the Semantic Versioning pattern (for example,. "**1.0.0**"). The JSON snippet, as shown in this example, provides just enough to make the service discoverable but does not do much else. Without this minimum configuration, the server would only return an HTTP-404 response for the service.

Define custom service endpoints

In the JSON mapping file, define custom service endpoints under an `operations` object. For example:

```
{
 "services": {
  "HelloService": {
   "version": "1.0.0",
   "operations": {
    "/Greeting": {},
    "/Message": {},
    "/": {}
   }
  }
 }
}
```

This registers the endpoints with the DOH class and makes them accessible as REST resources. Client applications access these resources using the full endpoint URI. For example:

- `http://localhost/CustomerApp/web/HelloService/Greeting`
- `http://localhost/CustomerApp/web/HelloService/Message`

Associate HTTP verbs with operation handlers

HTTP verbs

For each custom service endpoint, you associate HTTP verbs with operation handlers. You can set up operation handlers for all HTTP verbs that are supported by the WEB transport. These include:

- GET
- PUT
- POST
- DELETE
- HEAD
- OPTIONS
- TRACE
- PATCH

Specify each verb that you want to use as a JSON object under the service endpoint in the mapping file. For example:

```
{
 "services": {
  "HelloService": {
   "version": "1.0.0",
   "operations": {
    "/Greeting": {
     "GET": {},
     "PUT": {},
     "POST": {}
    },
    "/Message": {
     "GET": {},
     "PUT": {}
    },
    "/": {
     "GET": {}
    }
   }
  }
 }
}
```

Content types

Within each verb, define the MIME content type of the data that is returned by the endpoint. For example:

```
"/Greeting": {
 "GET": {
  "contentType": "application/json"
 }
}
```

All MIME types (`text/`, `image/`, `audio/`, `video/`, `application/`, `multipart/`) are supported.

Status codes

In addition to the content type, define the status code that is returned by the endpoint. For example:

```
"/Greeting": {
 "GET": {
  "contentType": "application/json",
  "statusCode": 200
 }
}
```

Operation handlers

Next, map each HTTP verb with one operation handler. The DOH class provides the following operation handlers:

- `void`, that returns only a status code
- `file`, that returns the contents of a file
- `entity`, that invokes an ABL class or procedure

To map an HTTP verb with an operation handler, nest the operation handler under the verb as shown in this example:

```
"/Greeting": {
 "GET": {
  "contentType": "application/json",
  "entity": {}
 },
 "PUT": {
  "contentType": "application/json",
  "entity": {}
 }
}
"/Message": {
 "GET": {
  "contentType": "application/json",
  "file": "filepath"
 }
}
"/": {
 "GET": {
  "contentType": "application/json",
  "void": null
 }
}
```

The `file` and `void` operation handlers are simple key-value pairs whereas the entity handler is a JSON object. This is because the entity handler requires further information (such as the ABL class or procedure name, the method that will be invoked, etc).

How to use operation handlers

Here is a detailed description of how you can use each of the DOH's operation handlers.

`void`
The `void` operation handler returns only the status code that is defined for the HTTP verb. This is useful in certain situations. For example, when you want to reserve an endpoint for future enhancements (via a `501` - Not Implemented status code) or when you want to use it to confirm that the server is alive (`204` - No Content).

Use the `void` handler as shown in this example:

```
"/CustomizedGreeting": {
 "GET": {
  "contentType": "application/json",
  "statusCode": 501,
  "void": null
 }
}
```

`file`
The `file` operation handler returns the contents of a file, as long as the contents of the file conform to a supported MIME type.

Note: For the `file` operation handler to work, the file must be placed within the web app directory, but not in the `WEB-INF` folder for security reasons. The `static` folder is the recommended location for it.

Since the web app may be deployed to different PAS for OpenEdge instances, as the ABL code gets moved from one environment to another (for example, from development to testing or from testing to production), the best practice is to reference the PAS for OpenEdge and web app directories using environment variables:

Directory	Environment variable
PAS for OpenEdge instance root	`${env.CATALINA_BASE}`
Web app directory (located in `webapps` in the PAS for OpenEdge instance root directory)	`${web.webapp}`

For example:

```
"/Message": {
 "GET": {
  "contentType": "application/json",
  "statusCode": 200,
  "file": "${env.CATALINA_BASE}/webapps/${web.webapp}/static/message.json"
 }
}
```

entity

You use the entity operation handler to map an HTTP verb with an ABL procedure or an ABL class method. At runtime, when the verb is used in an HTTP request to the service endpoint, the associated ABL procedure or class method gets executed.

Note: Although this operation handler is called entity, its use is not limited to ABL Business Entities. You can use it to reference any ABL class or procedure.

Here is a simple example of the entity handler:

```
"/Greeting": {
 "GET": {
  "entity": {
   "name": "Customer.Greeting",
   "function": "getPersonalizedGreeting",
   "arg": [
    {
     "ablName": "Cust-Name",
     "ablType": "CHARACTER",
     "ioMode": "INPUT",
     "msgElem": {
      "type": "query",
      "name": "CustomerName"
     }
    },
    {
     "ablName": "GreetingMessage",
     "ablType": "CHARACTER",
     "ioMode": "OUTPUT",
     "msgElem": {
      "type": "field",
      "name": "Message"
     }
    }
   ]
  }
 }
}
```

The name property identifies the ABL class (Customer.Greeting) that is to be referenced; the function property denotes the method (getPersonalizedGreeting) in the class that needs to be invoked. The ioMode property identifies the parameter type (INPUT, OUTPUT, etc).

The arg property (short for arguments) is an array of JSON objects. Each object maps to an input or output parameter that is required by the getPersonalizedGreeting method. The first three properties in each arg object (ablName, ablType, and ioMode) identify the ABL parameter and correspond to how the parameters are defined in the ABL method:

```
METHOD getPersonalizedGreeting(INPUT Cust-Name AS CHARACTER, OUTPUT GreetingMessage AS
 CHARACTER):
...
END.
```

The last property in each `arg` object--`msgElem`--defines how the parameter maps to a message element in the HTTP request or response. Parameters of the type INPUT retrieve their values from message elements in HTTP requests. Parameters of the type OUTPUT pass on their values (returned by the ABL method or procedure) to message elements in HTTP responses. The `msgElem` object therefore consists of two properties:

- The `type` of message element in an HTTP request or response. Examples include `query`, `header`, etc.
- The `name` of the message element in an HTTP request or response. Examples include `customer-id` (for a `query`), From, Host (for a `header`), etc.

At runtime, the DOH uses these property mappings to convert HTTP requests and responses into ABL objects. In some cases, such as for parameters that are of the type INPUT-OUTPUT, you may need to map HTTP requests as well as HTTP responses to the same parameter. In this case, you can use `msgElem` as an array as shown here:

```
"arg": [
 {
  "ablName": "CustomizedMessage",
  "ablType": "CHARACTER",
  "ioMode": "INPUT-OUTPUT",
  "msgElem": [
   {
    "type": "query",
    "name": "customer-name"
   },
   {
    "type": "field",
    "name": "message"
   }
  ]
 }
]
```

These examples illustrate a couple of different ways in which you can use the entity operation handler. The complete list of entity handler properties and the types of values that you can set for each property is described in the *Entity Handler properties* section.

Entity handler properties

You use the `entity` operation handler as a JSON object with the following properties:

- `type` — use this property to identify the type of ABL program. Possible values include `class` and `procedure`. If no value is set, the DOH uses `class` as the default value. If you are referencing a procedure, you must set the value to `procedure`.
- `name` — the name of the ABL class or persistent procedure, as is set in your application's PROPATH. If the ABL class is contained in a package, specify the full package name (for example, `Business.Customer`). If the ABL procedure file is in a folder in the ABL web app project, specify the relative path (for example, `Business/Customer`).
- `function` — the name of the ABL class method, procedure, or user-defined function that needs to be invoked.
- `arg` — a JSON array of objects to map the method or procedure's input and output parameters to message elements in the HTTP request or response. Each object in the array represents the mapping of a single parameter to HTTP message elements. Each parameter object contains the following properties:
  - `ablName` — the name of the ABL input or output parameter. If the parameter type is RETURN, as set in the `ioMode` property (see below), skip this property.

- `ablType` — the datatype of the ABL parameter. Possible values include `CHARACTER`, `LONGCHAR`, `INTEGER`, `INT64`, `DECIMAL`, `LOGICAL`, `ROWID`, `RECID`, `DATE`, `DATETIME`, `DATETIME-TZ`, `RAW`, `MEMPTR`, `DATASET`[1], `TABLE`[2], `BLOB`, `CLOB`, `EXTENT`[3], and `CLASS`[4].
- `ioMode` — the parameter type. Possible values include `INPUT`, `OUTPUT`, `INPUT-OUTPUT`, and `RETURN`.
- `msgElem` — a JSON object or array of objects that maps the ABL parameter to a message element in the HTTP request or response. Each `msgElem` object contains two properties--`type` and `name`. The value of the `name` property depends on the value of the `type` property. Specify values for `type` and `name` as described in this table.

Type	Name	When to use	Use for both requests and responses?
`none`	Leave the `name` property blank or use `null`.	• When the ABL parameter is an input type and there is no corresponding element in the HTTP request to map it to. Specifying `none` prompts the DOH to send an empty parameter to the ABL class method or ABL procedure. • When the ABL parameter is of an output type and you do not want to produce an HTTP message element from it.	Requests and Responses
`path`	Specify the name of the path parameter as defined in the WEB transport properties	When the ABL parameter is an input type and its value needs to be extracted from a parameter in the requested URL path. For example, assume that a client application makes a GET request to `http://localhost:8810/myapp/web/CustomerOrderService/customers/123` to retrieve customers records by customer ID (`123`). To set this up, set `path` as the msgElem `type` and `customerID` as the msgElem `name`. Edit the PAS for OpenEdge instance's `openedge.properties` file and add a handler for the path parameter in the web application's WEB transport properties section (that has a title similar to `oepas1.myapp.WEB`). Here is an example of the handler property to add: `handler2=OpenEdge.Web.DataObject.DataObjectHandler:/customers/{customerID}`	Requests only
`query`	Specify the name of the query parameter	When the ABL parameter is an input type and its value needs to be extracted from a query in the requested URL path. For example, assume that a client application makes a GET request to `http://localhost:8810/myapp/web/CustomerOrderService?id=123` to retrieve customers records by customer ID (`123`).	Requests only

[1] If you specify a `DATASET` as the `ablType`, you also need to define the dataset schema in the mapping file. To learn more, see the example dataset schema that follows the `msgElem` properties table.

[2] Use this to specify temp-tables. You also need to define the temp-table schema in the mapping file. To learn more, see the example dataset schema that follows the `msgElem` properties table below.

[3] If the parameter is an ABL array, use the syntax `<Datatype> EXTENT` (for example, `CHARACTER EXTENT`).

[4] If the parameter is an ABL object, use the syntax `CLASS <full.type.name>` (for example, `CLASS Business.Customer`)

Type	Name	When to use	Use for both requests and responses?
		To set this up, set `query` as the msgElem `type` and `id` as the msgElem `name`.	
`queryMap`	Leave the `name` property blank or use `null`	When the ABL parameter is an input type and the requested URL path contains a query of the type `OpenEdge.Core.Collections.IStringStringMap`. This is mostly useful for debugging.	Requests only
`header`	The name of the HTTP header. For complex headers with parameters, use the syntax `<headername>; <parameter>`	When the ABL parameter is an input, output, or input-output type, and its value needs to be either retrieved from an HTTP request header (for input operations) or passed on to an HTTP response header (for output operations).	Requests and responses
`headerSet`	Leave the `name` property blank or use `null`	When the ABL parameter is an input, output, or input-output type and its value is an array of HTTP header objects. This is mostly useful for debugging.	Requests and responses
`field`	The name of a root-level key in a JSON or multipart message	When the ABL parameter is an input, output, or input-output type, and the value needs to be retrieved from or passed on to a JSON or multi-part message in an HTTP request or response. You can only retrieve or pass the value of a key that is at the topmost level in the message hierarchy. For example, in the following JSON message, you can only retrieve the value of `message`, `status`, `date`, or `customer`. If you need to access a 'child' element (such as `text`, `target`, `id`, or `name` in this example), you must do so from within your ABL code. `{ "message": { "text": "Order number 213 was delivered." "target": "239888" }, "status": "sent", "date": "10-APRIL-2018", "customer": { "id": "1282", "name": "John Jones" } }`	Requests and responses
`cookie`	Specify the name of the browser cookie	When the ABL parameter is an input, output, or input-output type, and its value needs to be retrieved from or passed on to a single-value browser cookie.	Requests and responses
`statusCode`	Leave the `name` property blank or use `null`	When the ABL parameter is an output type, and the its value needs to be passed to the client application as an HTTP status code. Note that this overrides the `statusCode` property set for the HTTP verb.	Responses only

Type	Name	When to use	Use for both requests and responses?
`body`	Leave the `name` property blank or use `null`	When the ABL parameter is an input, output, or input-output type, and you need to either retrieve its value from the message body of the HTTP request or pass on the value from the ABL parameter to the message body of the HTTP response. While retrieving the value for an INPUT parameter, you can set a flag to strip the envelope from the HTTP request message. Similarly, in the case of an OUTPUT parameter, you can optionally set a flag to add the envelope to the parameter output before forming the HTTP response. To set these flags, add a an `options` property under the HTTP verb in the mapping file as shown in this example: `"/getPersonalizedMessage": { "GET": { "contentType": "application/json", "options": { "writeBI": true, "requestEnvelope" :true "responseEnvelope" :true "errorEnvelope" :true },`	Requests and responses
`request`	Leave the `name` property blank or use `null`	When the ABL parameter is an input type, and you want to populate its value with the complete HTTP request message, including the headers, body, etc.	Requests only
`httpMethod`	Specify the name of the HTTP verb	When you want to perform some type of debugging by passing the HTTP method name (GET, PUT, POST, etc) to an ABL method via an INPUT parameter.	Requests only
`constant`	Specify the value that you want to pass on to the ABL parameter	When you want to populate the value of an input parameter with the value of the `name` property. This is mostly useful for testing.	Requests only

Example of a dataset schema in the mapping file

If you specify `DATASET` in the `ablType` property, you must specify the dataset schema in the mapping file. Place the dataset or temp-table schema under a `schemas` object inside your *`ServiceName`* (`HelloService` in the following example), at the bottom of the JSON mapping file. Here is an example of a schema for a dataset that contains a temp-table:

```
"services": {
 "HelloService": {
  ....
  ....
  "schemas": {
      "dsCustomer" : {
      "ttCustomer": {
       "trackChanges": true,
       "indexes" : {
```

```
                "CustNum": {
                    "primary": true,
                    "unique": true,
                    "word": false,
                    "fields": ["CustNum"]
                },
                "Name": {
                    "primary": false,
                    "unique": false,
                    "word": false,
                    "fields": ["Name"]
                },
                "SalesRep": {
                    "primary": false,
                    "unique": false,
                    "word": false,
                    "fields": ["SalesRep"]
                }
        },
        "fields": [
                  {"name": "CustNum", "ablType": "INTEGER  ", "extent": null},
                  {"name": "Country", "ablType": "CHARACTER", "extent": null},
                  {"name": "Name", "ablType": "CHARACTER", "extent": null},
                  {"name": "Address", "ablType": "CHARACTER", "extent": null},
                  {"name": "Address2", "ablType": "CHARACTER", "extent": null},
                  {"name": "City", "ablType": "CHARACTER", "extent": null},
                  {"name": "State", "ablType": "CHARACTER", "extent": null},
                  {"name": "EmailAddress", "ablType": "CHARACTER", "extent": null}
        ]
       }
      }
     }
    }
  }
}
```

Related temp-tables in a dataset

If you have related temp-tables in a dataset and you want the response message to represent this relationship as nested JSON objects, you must define the relationship in the dataset schema. Here is the format (note the relationship in the `attr` object):

```
"schemas": {
   "<dataset-name>" : {
       "<temp-table-1>" : { /* definition */},
       "<temp-table-2>" : { /* definition */},
       "attr": {
            "relations": [
                 {
                    "parentTable": "<parent-temp-table-name>",
                    "childTable": "<child-temp-table-name>",
                    "fields": [
                         {
                           "parentField": "<parent-table-field-name>",
                           "childField": "<child-table-field-name>"
                         }
                     ],
                     "name": "<relation-name>",
                     "nested": true,
                     "active": true,
                     "recursive": false,
                     "reposition": false
                 }
            ]
       }
   }
}
```

Each of the relationship attributes (`name`, `nested`, `active`, `recursive`, and `reposition`) correspond to options that you can specify in the data-relation part of an ABL dataset definition.

Note: You can define multiple relationships in the `relations` JSON array. The relationships should be defined in the same order as in the ABL dataset definition.

Example

Consider a customer record that is related to multiple order records. Without a relationship defined (between the Customer and Order temp-tables) in the dataset schema, the Data Object Handler produces individual JSON objects in the response message:

```
{
    "dsEntityInfo":
        {
            "Customer":
                [
                    {
                     "CustNum": 1,
                     "CustName": "Lift Tours"
                    },
                ],
            "Order":
                [
                    {
                    "OrderNum": 100,
                    "OrderTotal": 1234.89,
                    "Customer_id": 1
                    },
                    {
                    "OrderNum": 150,
                    "OrderTotal": 999.99,
                    "Customer_id": 1
                    },
                 ]
        }
}
```

However, if the relationship is defined in the dataset schema, the Order records are nested under the Customer record in the response message, as shown in the example below.

Dataset schema example with related temp-tables

```
"schemas": {
   "dsEntityInfo" : {
       "Customer" : { /* definition */},
       "Order" : { /* definition */},
       "attr": {
            "relations": [
                {
                   "parentTable": "Customer",
                   "childTable": "Order",
                   "fields": [
                        {
                          "parentField": "CustNum",
                          "childField": "Customer_id"
                        }
                    ],
                    "name": "Cust-Order-Relation",
                    "nested": true,
                    "active": true,
                    "recursive": false,
                    "reposition": false
                }
            ]
       }
   }
}
```

Example output in response message

```
{
    "dsEntityInfo":
        {
            "Customer":
                [
                    {
                     "CustNum": 1,
                     "CustName": "Lift Tours"
                     "Order":
                     [
                       {
                       "OrderNum": 100,
                       "OrderTotal": 1234.89,
                       "Customer_id": 1
                       },
                       {
                       "OrderNum": 150,
                       "OrderTotal": 999.99,
                       "Customer_id": 1
                       }
                     ]
                    }
                ]
         }
}
```

OpenEdge DataServers

OpenEdge Release 11.7.4 Service Pack adds support for dataservers as described in the following sections:

Performance Improvement for inner join query having FIRST/LAST construct at any join level

Starting in this release with the DataServer for MS SQL Server, ABL inner join queries, having one FIRST or LAST criteria for a child table at any join level, are optimized to translate into a single join SQL statement that is evaluated on server. This could result in significant performance improvements for FIRST/LAST criteria specified at any level of an ABL inner join query.

An example of an ABL inner join query evaluated on the server is shown below. In the example, the FIRST construct is at the innermost join level, and the remaining buffers are joined using EACH statements. One join SQL statement is generated for this query, and it is evaluated on the server side by the MS SQL server database.

```
FOR EACH customer,
    EACH order OF customer,
    EACH orderline OF order,
    FIRST ITEM OF orderline:
    DISPLAY customer.custnum  order.custnum order.ordernum
    orderline.linenum item.itemnum.
END.
```

-nojoinhint startup parameter

Starting this release, the startup parameter `-nojoinhint` is introduced for the DataServer for Oracle. This parameter overrides the USE_MERGE default query-tuning hint for OUTER-JOIN queries. If only a few records are needed out of a large result set, the USE_MERGE default can be inefficient.

Disable the USE_MERGE hint by compiling your ABL application with the `-nojoinhint` client startup parameter.

The `-nojoinhint` parameter is for compile time only. It has no impact on run time.

For more information, see Join Hint (-nojoinhint) on page 16.

OpenEdge Management

This section describes the improved features in OpenEdge Management for OpenEdge that are included in the OpenEdge Release 11.7.4 Service Pack.

Create a PAS for OpenEdge instance

OpenEdge Management provides a sample PAS for OpenEdge instance (`oepas1`) with a default configuration. Each new instance of PAS for OpenEdge that you create uses the same default configuration, which you can edit.

OpenEdge Management monitors the PAS for OpenEdge installation and automatically creates resources if you use the `TCMAN` command to create a new PAS for OpenEdge instance. `TCMAN` is a command-line utility for managing and administering PAS for OpenEdge.

For more information about TCMAN, see the 'TCMAN Reference' in *PAS for OpenEdge Administration Tools Reference*.

To create a PAS for OpenEdge instance:

1. Click **Resources** > **New** > **Progress Application Server** in the OpenEdge Management console menu.

   The **Create new Progress Application Server** page appears.

2. Configure your Application Server as described in the following table:

Field	Description
AdminServer	Select an AdminServer for the PAS for OpenEdge instance. If you have created multiple remote AdminServers in OpenEdge Management or OpenEdge Explorer (**Options** > **Remote AdminServers**), this field is displayed for you to select an AdminServer.
Instance name	Specify a name for the PAS for OpenEdge instance. This is a mandatory field. **Note:** The PAS for OpenEdge instance name is case-sensitive. It can include any character except periods (.) or square brackets ([ ]). The name must be unique among all configured PAS for OpenEdge names.
Location	Select the location of the PAS for OpenEdge instance: • **Local**: OpenEdge Management has access to the machine where PAS for OpenEdge runs through an AdminServer. PAS for OpenEdge can be running either through a remote AdminServer or through an AdminServer in the same location as OpenEdge Management. • **Remote**: The PAS for OpenEdge instance is in a different location than OpenEdge Management, that is, the instance exists in a location without an AdminServer. When you select the location as **Remote**, you cannot not use OpenEdge Management to start or stop a PAS for OpenEdge instance. **Note:** OpenEdge Management monitors only the local installation of new PAS for OpenEdge instances. If you have installed PAS for OpenEdge as a stand-alone product and have not installed an OpenEdge product that includes an AdminServer, select **Remote** to monitor the instance. The OpenEdge manager web application, `oemanager`, must be installed on PAS for OpenEdge for you to use the PAS for OpenEdge instance from a remote location.
Security model	Select a security model: • **Production**: The *oemanager* and *manager* Web applications need to be deployed separately. Make configuration changes to adjust security settings to ensure your application operates correctly in a production instance. • **Developer**: The *oemanager* and *manager* Web applications are deployed by default.

Field	Description
	Development mode minimizes or eliminates any accessibility restrictions, so a developer can quickly use the product “out of the box” to develop, test, and debug applications.
URL	Specify the URL of the PAS for OpenEdge instance if PAS for OpenEdge is running remotely. The format of the URL must be `http(s)://server:port/oemanager`. **Note:** For a local instance, OpenEdge Management or OpenEdge Explorer constructs the URL using localhost and port for secure communication. If you want to use HTTP to monitor the instance, you must specify the URL manually. For using secure communication, you must install all the required certificates in `$DLC/certs`.
No host verification for SSL	Select the check box if you do not want to validate the hostname in the URL of the PAS for OpenEdge instance that is enabled for secure communication.
Instance directory	Specify the name of the directory where the PAS for OpenEdge instance resides. This is an optional field. **Note:** If you do not provide the name of the instance directory, the PAS for OpenEdge instance is created in the working directory set during OpenEdge installation. The name of the instance is used as the name of the directory where the PAS for OpenEdge installation is created. The directory must not already exist (either when provided, or when using the default location).
HTTP port	Specify an unused port number associated with the PAS for OpenEdge HTTP. By default, this is set to 8080. **Note:** Each new PAS for OpenEdge instance that you create uses the default configuration. However, the port number must be unique for each PAS for OpenEdge instance for the instance to operate properly. If you specify a port number that is used by another PAS for OpenEdge instance, you are prompted to confirm whether you want to use the port number.
HTTPS port	Specify an unused port number associated with the PAS for OpenEdge HTTPS. By default, this is set to 8443. **Note:** If you specify a port number that is used by another PAS for OpenEdge instance, you are prompted to confirm whether you want to use the port number.
Shutdown port	Specify an unused port number for shutdown. If you are creating a PAS for OpenEdge instance on a Windows machine, this is a mandatory field. **Note:** If you specify a port number that is used by another PAS for OpenEdge instance, you are prompted to confirm whether you want to use the port number.

Field	Description
Autostart	Set the **Autostart** option to automatically start PAS for OpenEdge every time you start the AdminServer.
Tomcat manager login	Specify the login ID of the Tomcat Web application that hosts the PAS for OpenEdge instance. If you are using the Tomcat Web server shipped with OpenEdge, then the default login ID is `tomcat`. You can also specify your own login ID other than the default ID, and it is stored in the `tomcat-users.xml` file.
Tomcat manager password	Specify the password of the Tomcat Web application that hosts the PAS for OpenEdge instance. If you are using the Tomcat Web server shipped with OpenEdge, then the default password is `tomcat`. You can also specify your own login password other than the default password, and it is stored in the `tomcat-users.xml` file.

3. Click **Save** to complete creating the PAS for OpenEdge instance.

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

Note:

The oemanager and manager web applications are installed depending on the following combinations:

License	Security Model	Result
Progress Dev AS for OE	Developer	oemanager and manager are installed.
Progress Dev AS for OE	Production	oemanager and manager are not installed.
Progress Prod AS for OE	Developer	oemanager and manager are not installed.
Progress Prod AS for OE	Production	oemanager and manager are not installed.
Progress Dev AS for OE and Progress Prod AS for OE	Developer	oemanager and manager are installed.
Progress Dev AS for OE and Progress Prod AS for OE	Production	oemanager and manager are not installed.

Manage PAS for OpenEdge instance

You can install and manage ABL apps, Web apps and Web app services in PAS for OpenEdge using the **ABL APPLICATIONS**, **WEB APPLICATIONS** and **WEB APP SERVICES** tabs. All the related command and controls for ABL apps, Web apps and Web app services are internally performed using the *tcman* utility. Consequently, these operations are available only for the instances that are hosted locally, and not for the remotely hosted ABL apps, Web apps and Web app services where the *tcman* utility is inaccessible to OpenEdge Management and OpenEdge Explorer. Therefore, to manage remote ABL apps, Web apps and Web app services instances, you must use the *tcman* utility in the remote location. For more information on managing ABL apps, Web apps and Web app service instances using the *tcman* utility, see the TCMAN reference sections in your OpenEdge documentation.

Note: You must install only the ABL apps, Web apps and Web app services that you want to manage. The OpenEdge manager web application, `oemanager`, must be installed on PAS for OpenEdge for you to manage OpenEdge ABL apps, Web apps and Web app services when the PAS for OpenEdge instance is running. The Tomcat manager Web application, `manager`, must be installed to deploy and undeploy ABL apps, Web apps and Web app services when the PAS for OpenEdge instance is running. You may choose not to install these Web applications to improve security, but in such a scenario, OpenEdge Management or OpenEdge Explorer cannot manage the ABL apps, Web apps and Web app services.

The following sections discuss the different commands and controls used to configure PAS for OpenEdge. These are all the different options under a PAS for OpenEdge instance page.

You can perform the following functions from the **Progress Application Server: *`instance name`*** page:

- Change PAS for OpenEdge instance control settings on page 36
- Install other Web applications shipped with PAS for OpenEdge on page 36
- Manage ABL applications on page 37
- Manage Web applications on page 38
- Manage Web app services on page 39
- Configure a PAS for OpenEdge instance on page 40
- Register a PAS for OpenEdge instance as a service on page 41

The **Progress Application Server: *`instance name`*** page displays five of the most recent alerts. You can clear an individual alert or click **Clear all alerts** to clear all the alerts from the page. When there are more than five alerts, the page displays a **See complete list** button to open the **Alerts** page, which displays all alerts. For more information about alert features, see *OpenEdge Management: Alerts Guide and Reference*.

To view the graphs and charts pertaining to the CPU usage, physical, and virtual memory for PAS from Resources for a selected PAS instance, navigate to **Resources > Go to Resources > Progress Application Server > *`PAS instance`*** and click **Graphs/Charts**. Using the options on the top-right corner of the graph, you can select a time period, ranging from 2 hours to 2 weeks, to display the data collected for that period of time, reload the graph, and select different forms of graphical interpretation.

Perform operations on PAS for OpenEdge instances

The ABL applications page view appears as the default landing page for a specific PAS for OpenEdge instance.

This page displays the following tabs to select and perform required operations on:

- **ABL Applications** — Manage ABL applications on page 37
- **Web Applications** — Manage Web applications on page 38
- **Web App Services** — Manage Web app services on page 39

You can also navigate to the following pages:

Page Name	Description
Instance Control	Start or stop a PAS for OpenEdge instance.
Monitoring Plans	Define monitoring plans to alert on PAS for OpenEdge instance conditions.
Configuration	Display the Progress file configuration associated with the instance.
Log File	Perform view and examine operations on log files. • **Catalina Log File Monitor** — Log file on which to alert • **Catalina Log File Viewer** — To examine the log file • **Host Manager Log File Monitor** — Log file on which to alert • **Host Manager Log File Viewer** — To examine the log file
Register/Unregister as Service	Register or unregister the instance as a Windows service (Windows only).
Charts/Graphs	Displays charts and graphs on CPU, Memory and Virtual Memory usage for the instance on Tomcat.

Change PAS for OpenEdge instance control settings

From the **Progress Application Server: <instance name>** details page, you can access the **Instance Control** page. The **Instance Control** page provides information about the current PAS for OpenEdge instance, PAS for OpenEdge status, and PAS for OpenEdge agent settings, and allows you to start and stop PAS for OpenEdge.

You can perform the following from the **Instance Control** page of a PAS for OpenEdge instance:

- Stop PAS for OpenEdge by clicking **Stop Progress Application Server**. Click **Yes** in the confirmation dialog that appears. The status is updated to **Not Running**. To restart PAS for OpenEdge, click **Start Progress Application Server**.

  The PAS for OpenEdge agent works only if PAS for OpenEdge is running. Therefore, when you stop PAS for OpenEdge, the monitoring agent status automatically changes to **Not Running**.

- Change the status of a PAS for OpenEdge monitor from **Enabled** to **Disabled** by clicking **Edit** and then clearing the **Enabled** option.

  Even after disabling PAS for OpenEdge monitors, PAS for OpenEdge still appears under **Progress Application Server** in the **Resources** list. However, PAS for OpenEdge is no longer being monitored by OpenEdge Management.

Install other Web applications shipped with PAS for OpenEdge

You can install the Web applications that are shipped with PAS for OpenEdge from the **Progress Application Server: <instance name>** page. This option is beneficial if you have not installed these Web applications while creating the PAS for OpenEdge instance or if you have previously undeployed these Web applications.

To install other Web applications shipped with PAS for OpenEdge:

1. Click **Resources** > **Go to Resources** in the OpenEdge Management console menu.

   All resources managed by your console appear in the grid frame.

2. Filter or search for, and select the required PAS for OpenEdge instance.

For example, type the default PAS for OpenEdge instance name, `oepas1`, in the **Filter** field or select **Progress Application Server** as the selection in the **Type** drop-down menu and select the PAS instance.

The **Progress Application Server: <instance name>** page appears.

3. In the **Command and control** section of the page, click **Extras**. The **Extras** page appears.

   For the Web applications that are already installed on the PAS for OpenEdge, a notification is displayed.

4. If available, select the following options to install the Web applications:

   - Select the **Install Progress Application Server OpenEdge manager web application** option to install the OpenEdge manager web application, `oemanager`.
   - Select the **Install OpenEdge ABL web application** option to install the OpenEdge ABL Web application, `oeabl`.
   - Select the **Install Tomcat manager web application** option to install the Apache Tomcat manager Web application, `manager`.
   - Select the **Install Tomcat host manager web application** option to install the Apache Tomcat host manager Web application, `host-manager`.

5. Click:

   - **Save** to install the selected Web applications on the PAS for OpenEdge instance.
   - **Cancel** to cancel any changes.

Manage ABL applications

To manage ABL applications, ensure that the PAS for OpenEdge instance is running. From the OpenEdge Management menu, select **Resources-> Progress Application Server**. Filter or search for and select the required PAS for OpenEdge instance.

The **ABL APPLICATIONS** page appears by default, listing all the existing ABL applications managed by PAS for OpenEdge.

Perform any of the following operations:

Operation	Action Description
Refresh	Click to refresh the ABL application grid.
Deploy	To deploy an ABL application. Click the **Deploy** tab. The **Deploy ABL Application** wizard appears. **1.** In the **Select application .WAR** field, perform one of the following actions to upload an ABL application. **Note:** Ensure that the selected file is an ABL application. • **Browse** to locate and select an ABL application .war file. **2.** Type a name in the **ABL application name** field. Select the application name from the auto suggest list that appears. The name must contain only alphanumeric characters. It must not contain blank spaces or special characters, except underscore (“_”) and hyphen (“-”). **3.** Type a name for the associated web application in the **Web application name** field **4.** **Deploy** the application on PAS for OpenEdge. The OpenEdge ABL Web application is deployed.
Undeploy	This operation is disabled by default and is enabled when an ABL application is selected to undeploy an ABL application on PAS for OpenEdge. Choose multiple ABL applications to perform this operation in bulk.

Manage Web applications

To manage your Web applications on the PAS for OpenEdge:

1. Click **Resources** > **Go to Resources** in the OpenEdge Management console menu.

   All resources managed by your console appear in the grid frame.

2. Filter or search for, and select the required PAS for OpenEdge instance.

   For example, type the default PAS for OpenEdge instance name, `oepas1`, in the **Filter** field or select **Progress Application Server** as the selection in the **Type** drop-down menu and select the PAS instance.

   The **Progress Application Server: <instance name>** page appears.

   The **ABL APPLICATIONS** page appears by default, listing all the existing ABL applications managed by PAS for OpenEdge.

   Select the **WEB APPLICATIONS** tab. The Web applications grid appears, listing all the existing Web applications managed by PAS for OpenEdge.

Operation	Perform
Deploy	To deploy a Web application. Click **Deploy**. The **Deploy web application** dialog appears. **1.** In the **Select application .war file** field, perform one of the following actions to upload a Web application. **Note:** Ensure that the selected file is an Web application. • Type a file name to select the Web application .war file from the extras folder. The auto suggest list appears as you type the name. You can also give a new name for the Web application. • **Browse** to locate and select an application .war file. • Type a name for the selected application in the **Web application name** field. The name must contain only alphanumeric characters. It must not contain blank spaces or special characters, except underscore (“_”) and hyphen (“-”). • Click **Deploy** to deploy the application on PAS for OpenEdge. The OpenEdge Web application is deployed.
Start	This operation is disabled by default and is enabled when there is a Web application(s) available with **Status** *stopped*. Select a Web application(s) and click **Start** to start a Web application. Choose multiple Web applications to perform this operation in bulk.
Stop	This operation is disabled by default and is enabled when there is a Web application(s) available with **Status** *running*. Select a Web application(s) and click **Stop** to stop a Web application. Choose multiple Web applications to perform this operation in bulk. This option makes the Web application and all its services unavailable for requests. **Note:** The Web application is automatically restarted when the PAS for OpenEdge instance is started.
Undeploy	Select a Web application and click **Undeploy** to undeploy a Web application from PAS for OpenEdge. Choose multiple Web applications to perform this operation in bulk. This option stops the Web application and physically removes it from PAS for OpenEdge.

Manage Web app services

To manage Web App Services, ensure that the PAS for OpenEdge instance is running. From the OpenEdge Management menu, select **Resources-> Progress Application Server**. Filter or search for and select the required PAS for OpenEdge instance.

The **ABL APPLICATIONS** page appears by default, listing all the existing ABL applications managed by PAS for OpenEdge.

Select the **WEB APP SERVICES** tab. The Web application services grid displays a list of deployed Web application services, if available. The **_oepingService** is the default REST **Service name** for any Web application. Perform any of the following operations:

Operation	Action Description
Refresh	Click to refresh the Web application services grid.
Deploy	To deploy a Web application service of type REST or SOAP .Click the **Deploy** tab. The **Deploy web service** wizard appears. **1.** In the **Select transport service file** field, perform one of the following actions to upload a Web application service. • **Browse** to locate and select a `.paar`, `.zip` or `.wsm` file. • In the **Web application name** field, search and select the associated Web application name for the Web application service. **Note:** This field appears only when a `.paar` or `.zip` is selected. **2.** Type a name for the Web service in the **Web service name**. **3.** **Deploy** the Web service application on PAS for OpenEdge. The OpenEdge Web service application is deployed.
Undeploy	This operation is disabled by default and is enabled when a Web application service is selected. To undeploy a Web application service on PAS for OpenEdge. Choose multiple Web application services to perform this operation in bulk.
Disable	This operation is disabled by default and is enabled when there is a Web application service(s) available with **State** *ENABLED*. To disable a Web application service. Choose multiple Web application services to perform this operation in bulk.
Enable	This operation is disabled by default and is enabled when there is a Web application service(s) available with **State** *DISABLED*. To enable a Web application service. Choose multiple Web application services to perform this operation in bulk.

Configure a PAS for OpenEdge instance

You can set the configuration properties of a PAS for OpenEdge instance using the **Configure Progress Application Server Properties** page.

To set PAS for OpenEdge configuration properties:

1. Click **Resources** > **Go to Resources** in the OpenEdge Management console menu.

   All resources managed by your console appear in the grid frame.

2. Filter or search for, and select the required PAS for OpenEdge instance.

   For example, type the default PAS for OpenEdge instance name, `oepas1`, in the **Filter** field or select **Progress Application Server** as the selection in the **Type** drop-down menu and select the PAS instance.

   The **Progress Application Server: <instance name>** page appears.

3. In the **Command and Control** section, click **Configuration**.

   The **Configure Progress Application Server Properties** page appears with the following tabs:

- **Connection Configuration** — Use this tab to configure the properties required for the OpenEdge Management to communicate with the Pacific Application Server.
- **Server Configuration** — Use this tab and the other tabs under it to configure the server properties that are stored in the `catalina.properties` and `appserver.properties` files.

Register a PAS for OpenEdge instance as a service

Just like you can manage the AdminServer (

Progress OpenEdge AdminServer Service

<version number>) as a service in the Services view of Microsoft Management Console, you can register and manage a PAS for OpenEdge instance as a service as well.

Note: Registering a PAS for OpenEdge instance as a service is applicable only for Windows platforms.

You cannot register a *development* instance of PAS for OpenEdge as a service. However, if a development instance was registered as a service in previous release of OpenEdge, you can unregister that instance as a service.

To register a PAS for OpenEdge instance as a service:

1. Click **Resources** > **Go to Resources** in the OpenEdge Management console menu.

   All resources managed by your console appear in the grid frame.

2. Filter or search for, and select the required PAS for OpenEdge instance.

   For example, type a PAS for OpenEdge instance name in the **Filter** field or select **Progress Application Server** as the selection in the **Type** drop-down menu and select the PAS instance.

   The **Progress Application Server: <instance name>** page appears.

3. In the **Command and control** section of the page, click **Register as service**.

   The **Register PAS as Service** page appears.

4. Click **Register as Service** to register the PAS for OpenEdge as a service.

   You can now start or stop the PAS for OpenEdge instance in the Services view of Microsoft Management Console. Note that the **Start** and **Stop** buttons in the **Progress Application Server: <instance name>** page changes to **Start as windows service** and **Stop as windows service**, respectively.

5. Optionally, you can select **Unregister as service** on the **Progress Application Server: <instance name>** page to unregister the PAS for OpenEdge instance as a service.

Progress Application Server for OpenEdge

This section describes the new features in PAS for OpenEdge and content updates that are included in the OpenEdge Release 11.7.4 Service Pack.

Improve performance with new tuning and sizing guides

The PAS for OpenEdge Machine Sizing Guide and PAS for OpenEdge Tuning Guide are intended for administrators who want to best performance out of their PAS for OpenEdge machines in production.

Progress Application Server for Openedge: Machine Sizing Guide

The PAS for OpenEdge Machine Sizing Guide describes how to run performance tests on your PAS for OpenEdge application, and provides a framework for sizing your machines appropriately for use in production. The document is hosted on the Progress Communities OpenEdge General group. The PAS for OpenEdge ATM Benchmark Test files are also available on Progress Communities if you want to run the machine sizing test that was used at Progress.

Progress Application Server for OpenEdge: Tuning Guide for 11.7

The PAS for OpenEdge Tuning Guide for 11.7 describes the configuration controls that can be adjusted in order to achieve optimal performance for an ABL application running in OpenEdge 11.7.

Manage and monitor an instance

Manage and monitor an instance with these enhancements to PAS for OpenEdge.

- Use Swagger UI to explore management REST APIs on page 42
- Use OEJMX to manage and monitor an instance on page 44
- Automate lifecycle event scripts on page 53
- New content for management REST APIs on page 54

Use Swagger UI to explore management REST APIs

System administrators can use Swagger UI to explore the underlying management APIs for a running PAS for OpenEdge instance. Part of the larger OpenAPI Initiative, Swagger UI is an open source tool that generates an intuitive web page to explore and test REST APIs. Details are pulled directly from the underlying annotations ensuring that the documentation is always current. Test capabilities allow real-time access to the APIs. System administrators can view metrics, identify deployed applications, stop agents, and do much more through this interactive documentation.

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

In OpenEdge 12.0 and 12.1, Swagger UI is disabled by default on development instances. In OpenEdge 12.2, Swagger UI is enabled by default on development instances. Swagger UI should not be used in production.

See also:

- Enable Swagger UI for management REST API access on page 42
- Explore and test management REST APIs on page 43

Enable Swagger UI for management REST API access

Management REST API access requires `oemanager.war` be deployed. Create development instances with the `-f` option to deploy `oemanager.war`. Secure the instance by using the `-m` *`uid:password`* option to change the username and password from the default values. Start the instance at least once to create the `oemanager` folder structure containing the necessary configuration files.

Note: This topic applies to Swagger UI in OpenEdge 12.0 and 12.1. In OpenEdge 12.2, Swagger UI is enabled by default on development instances. Swagger UI should never be enabled in production.

Access to Swagger UI for management REST APIs is disabled by default. Direct access to management APIs through Swagger UI is considered a security risk. Explicit authorization is required. Limit access to the administrator roles within the organization. Restart the server to apply the security changes. To enable Swagger UI:

1. Edit the *instance-path*\webapps\oemanager\WEB-INF\oemgrSecurity-container.xml security file.
2. Find the intercept URL for the interactive documentation, by searching the file for `<intercept-url pattern="/doc/**"`.
3. Uncomment the line containing `<intercept-url pattern="/doc/**` that authorizes specific roles to use the interactive documentation.
4. Save the file.
5. Restart the server to apply the changes:

   ```
   pasman pasoestart -restart -I instance-name
   ```

6. Enter `http://host:port/oemanager/` to show the management REST API documentation, confirming that authorization was correctly defined. Login credentials are required to ensure only administrators have access.

Explore and test management REST APIs

Access to Swagger UI is disabled by default. To enable this functionality, see Enable Swagger UI for management REST API access on page 42.

In this example, explore the **SessionManager** APIs and get all the available applications for an instance. Based on the response, use the **Agent Manager** getAgent API to get the PID for the agent currently running a named application. To explore and test a REST API:

1. Enter `http://host:port/oemanager/` in a browser to access interactive documentation, and provide the appropriate login credentials when prompted.
2. Verify that `oemanager` is the selected **Server**.
3. Scroll to **SessionManager** and expand the list of available APIs.
4. Click on **GET** for the `/applications - Get All Applications` to expand the description based on the underlying annotations.
5. Click **Try it out** to supply request parameters and click **Execute** to retrieve a list of available applications.

   The response body is JSON formatted and the `applicationName` element provids details about the deployed applications, including which transports are enabled for the application and the version information.
6. Take note of the `applicationName` for use as the request body to find the Agent running the application.
7. Expand the **Agent Manager** section.
8. Click **/applications/{appName}/agents Get Agents**.
9. Click **Try It Out**.
10. Enter the `applicationName` returned in the earlier step into the Parameters **AppName** field.
11. Click **Execute**.

    The response returns a **PID**, for use with API requests.

For more information, see Monitor PAS for OpenEdge.

Use OEJMX to manage and monitor an instance

You can monitor and manage applications using the Java Monitoring and Management Console (JConsole) and the OpenEdge Manager application (`oemanager`). You can also monitor and manage instances through scripts using the OpenEdge JMX command-line utility, OEJMX, to connect to a local PAS for OpenEdge instance.

Based on Java Management Extensions (JMX), the `OEJMX` utility connects to an instance using Java objects called Managed Beans (MBeans). One or more MBeans instrument a resource using attributes and operations. This direct access to the underlying MBeans provides administrators with the freedom to write scripts to monitor and manage applications in ways that may not be available within OpenEdge Manager or JConsole.

For the OEJMX command syntax, see OEJMX utility.

For usage information, see the following topics:

- List available MBeans to locate AgentManager details on page 44
- Use an OEJMX query to get agent information on page 46
- Use an OEJMX query to get session metrics
- Common OEJMX queries on page 50

List available MBeans to locate AgentManager details

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

To list the available MBeans and locate AgentManager details:

1. From the target PAS for OpenEdge instance's `bin` directory, start the instance, for example:

```
proenv>tcman pasoestart
```

2. Run `oejmx -C` to create a list of available MBeans, for example:

```
proenv>oejmx -C
```

   Output is sent to the default location,`/temp/beanInfo-`*`timestamp`*`.txt`.

3. Open the output file and search for the `AgentManager` details. The AgentManager MBean lists the available operations that can be used to get agent information, status, and other metrics. The JMX queries are displayed on the lines that begin with `{"O":"PASOE:type=OEManager,name=AgentManager"` as shown in the sample AgentManager MBean operations output:

```
**************** PASOE:type=OEManager,name=AgentManager
Operation: getAgents (String appName) -->  javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getAgents","<appName>"]}
Operation: refreshAgents (String appName) -->  javax.management.openmbean.CompositeData
```

```
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["refreshAgents","<appName>"]}
Operation: refreshAgent (String id) -->  javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["refreshAgent","<id>"]}
Operation: getAgentStatus (String id) -->  javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getAgentStatus","<id>"]}
Operation: flushDeferredLog (String appName, String agentId) -->
javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["flushDeferredLog","<appName>","<agentId>"]}
Operation: resetDeferredLog (String appName, String agentId) -->
javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["resetDeferredLog","<appName>","<agentId>"]}
Operation: clearAgentStatData (String id) -->  Boolean
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["clearAgentStatData","<id>"]}
Operation: getAgentMetrics (String id) -->  javax.management.openmbean.CompositeData

  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getAgentMetrics","<id>"]}
Operation: getThreadMetrics (String id) -->  javax.management.openmbean.CompositeData

  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getThreadMetrics","<id>"]}
Operation: getSessionMetrics (String id) -->  javax.management.openmbean.CompositeData

  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getSessionMetrics","<id>"]}
Operation: getConnectionMetrics (String id) -->
javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getConnectionMetrics","<id>"]}
Operation: getRequestMetrics (String id) -->  javax.management.openmbean.CompositeData

  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getRequestMetrics","<id>"]}
Operation: getAgentStacks (String id) -->  javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getAgentStacks","<id>"]}
Operation: getSessionStacks (String agentID, String sessionID) -->
javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["getSessionStacks","<agentID>","<sessionID>"]}
Operation: stopAgent (String agentID, Integer waitToFinish, Integer waitAfterStop)
-->  boolean

{"O":"PASOE:type=OEManager,name=AgentManager","M":["stopAgent","<agentID>",<waitToFinish>,<waitAfterStop>]}
Operation: getDynamicABLSessionLimit (String appName, String id) -->
javax.management.openmbean.CompositeData[]

{"O":"PASOE:type=OEManager,name=AgentManager","M":["getDynamicABLSessionLimit","<appName>","<id>"]}
Operation: resetDynamicABLSessionLimit (String appName, String id) -->
javax.management.openmbean.CompositeData[]

{"O":"PASOE:type=OEManager,name=AgentManager","M":["resetDynamicABLSessionLimit","<appName>","<id>"]}
Operation: setDynamicABLSessionLimit (String appName, String id, Integer sessionLimit)
 -->  javax.management.openmbean.CompositeData[]

{"O":"PASOE:type=OEManager,name=AgentManager","M":["setDynamicABLSessionLimit","<appName>","<id>",<sessionLimit>]}
Operation: terminateABLSession (String agentID, Integer agentSessionID, Integer
terminateOption) -->  boolean

{"O":"PASOE:type=OEManager,name=AgentManager","M":["terminateABLSession","<agentID>",<agentSessionID>,<terminateOption>]}
Operation: trackABLObjects (String agentID, Boolean switch) -->  Boolean

{"O":"PASOE:type=OEManager,name=AgentManager","M":["trackABLObjects","<agentID>",<switch>]}
Operation: trackingABLObjects (String agentID) -->  boolean
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["trackingABLObjects","<agentID>"]}
Operation: getABLObjectsReport (String agentID) -->
javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["getABLObjectsReport","<agentID>"]}
Operation: getABLSessionObjectsReport (String agentID, Integer agentSessionID) -->
 javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["getABLSessionObjectsReport","<agentID>",<agentSessionID>]}
```

```
Operation: trackMemoryUse (String agentID, Boolean switch) -->  Boolean

{"O":"PASOE:type=OEManager,name=AgentManager","M":["trackMemoryUse","<agentID>",<switch>]}
Operation: trackingMemoryUse (String agentID) -->  boolean
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["trackingMemoryUse","<agentID>"]}
Operation: getMemoryUseReport (String agentID) -->
javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getMemoryUseReport","<agentID>"]}
Operation: getMemoryUseSessionReport (String agentID, Integer agentSessionID) -->
javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["getMemoryUseSessionReport","<agentID>",<agentSessionID>]}
Operation: initiateDebugger (String id, String connectString) -->  boolean

{"O":"PASOE:type=OEManager,name=AgentManager","M":["initiateDebugger","<id>","<connectString>"]}
Operation: debugTest (String agentID, String operation, String modifiers, Integer
iparm, String cparm) -->  javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["debugTest","<agentID>","<operation>","<modifiers>",<iparm>,"<cparm>"]}
Operation: pushProfilerData (String id, String url, Integer requestCount, String
profilerSettings) -->  javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["pushProfilerData","<id>","<url>",<requestCount>,"<profilerSettings>"]}
Operation: getProfilerSettings (String id) -->
javax.management.openmbean.CompositeData
  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getProfilerSettings","<id>"]}
Operation: dynamicPropertiesUpdate (String id, String propertiesString) -->  boolean


{"O":"PASOE:type=OEManager,name=AgentManager","M":["dynamicPropertiesUpdate","<id>","<propertiesString>"]}
Operation: getProperties (String appName) -->  javax.management.openmbean.CompositeData

  {"O":"PASOE:type=OEManager,name=AgentManager","M":["getProperties","<appName>"]}
Operation: setProperties (String appName, String PropertyName, String PropertyValue)
 -->  javax.management.openmbean.CompositeData

{"O":"PASOE:type=OEManager,name=AgentManager","M":["setProperties","<appName>","<PropertyName>","<PropertyValue>"]}
```

For example, the `stopAgent` operation could be used to stop an agent, and the `terminateABLSession` operation could be used to stop a particular session. These operations are described in the OEJMX Query Reference.

Some JMX queries require you to replace the variables with information obtained by running a `getAgents` query, or by providing other data such as application name, property name, or URL. Writing the `getAgents` query is described in Use an OEJMX query to get agent information on page 46.

Note: Detailed information about available MBeans can be found by using JConsole. For more information, see JConsole and JMX.

With a list of available MBeans, you can Use an OEJMX query to get agent information on page 46.

Use an OEJMX query to get agent information

Use OEJMX with the `-Q` *`filepathname`* parameter to use a custom query file. A query file can have multiple queries. For more information about the OEJMX command syntax, see OEJMX utility on page 51.

Each JMX query is a JSON string with an MBean identifier and attribute or operation information. The query format is:

- **`O`**—ObjectName is a JsonObject
- **`M`**—Operation is a JsonArray

- **A**—(optional) Attribute is a JsonObject

An example is the JMX query for getting agent information:

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

In List available MBeans to locate AgentManager details on page 44, steps are provided to list the available MBeans. From that list, the operations and attributes for the AgentManager are identified. To collect information about the agents, create a custom OEJMX query file using the `getAgents` API for the default `oepas1` instance.

To write a custom OEJMX query:

1. Create a new file in your PAS for OpenEdge instance's `bin\jmxqueries` directory called `getAgents.qry`.
2. Add the following line to the `getAgents.qry` file:

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

   This line contains a JMX query to access the **AgentManager** MBean, and specifically the `getAgents` JMX API. This provides information about the running agents in the PAS for OpenEdge instance, which can be used to get thread and session information, status, and other metrics.

   The query returns the agent information for the default `oepas1` instance. If you are running this query on a PAS for OpenEdge instance with a different name, change `oepas1` to the name of the current instance or ABL application that you want to get agent information about.

   If you want agent information for all running ABL applications on the PAS for OpenEdge instance, you can specify `APP_NAME` instead of `oepas1` (or other ABL application name). `APP_NAME` is a supported token that returns information for all ABL applications on the PAS for OpenEdge instance. More information about supported tokens is described inCommon OEJMX queries on page 50.

   Note: A query must be a single line that does not contain embedded spaces. A query file can contain multiple queries. The best practice is to use explicit names over tokens to guarantee the intended results. In this example, `oepas1` replaces the *APP_NAME* token.

3. Navigate to the `bin` directory of the instance, for example:

   ```
   proenv>cd %WRKDIR%\oepas1\bin
   ```

4. Ensure the instance is running by executing the following command:

   ```
   proenv>tcman pasoestart -restart
   ```

5. Run the OEJMX getAgents query that you created in step one, for example:

   ```
   proenv>oejmx -Q jmxqueries\getAgents.qry
   ```

The default output location is *instance_name*/temp/*queryname*-out-*yyMMdd-HHmm*.txt.

6. Review the output getAgents-out*yyMMdd-HHmm*.txt file in the *instance_name*/temp directory to see the query response, for example:

```
Query: Line 1. PASOE:type=OEManager,name=AgentManager, getAgents(oepas1)
Result:
{"getAgents":{"agents":[{"agentId":"b4XlcFFOR_uh68bStvHe1Q","pid":"26896","state":"AVAILABLE"}]}}
```

Note: The output shows the original query on line 1 and the results on the next line. The original query message can be suppressed by running the OEJMX query with the -R option, for example:

```
proenv>oejmx -Q jmxqueries\getAgents.qry -R
```

For more information about the OEJMX command syntax, see OEJMX utility on page 51.

The output of the getAgents query provides the Agent ID and PID for the running ABL application agents. You can use this information to gather more information about your agent, such as threads and session metrics. For more information, see Use an OEJMX query to get session metrics and Common OEJMX queries.

Get agent information

A running PAS for OpenEdge instance must be available. Review Use an OEJMX query to get agent information on page 46 to get the available agent.

To get agent information:

1. Run the Proenv utility.
2. Go to the *instance-name*/bin directory.
3. Execute the query using oejmx -Q *Filepath*/getAgents.qry

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

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

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

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

For more information about the OEJMX utility, see the OEJMX utility on page 51.

Generate a stack trace

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

1. Write a query using `getSessionMetrics`, for example:

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

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

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

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

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

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

   This collects the `getSessionStacks` details.

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

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

The output displays the stack trace listing files that were used by the session.

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

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

From more information, see the Common OEJMX queries on page 50 .

Common OEJMX queries

One of the fundamental OEJMX queries is to request agents for an ABL application. This query provides you with information about the multi-session agents running in your application. You can add these queries to build out a multi-line query file to get information about your PAS for OpenEdge instance.

Get agents from an ABL application

When to use: To identify agents in the ABL application.

Query:

```
# Tokens: APP_NAME
# Get agents in an ABL application
{"O":"PASOE:type=OEManager,name=AgentManager","M":["getAgents","oepas1"]}
```

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

Result:

The output provides details about the agents including agent ID, PID, and state information. The agent ID can be used in other OEJMX queries such as to refresh an agent in an ABL application.

```
{
  "getAgents": {
    "agents": [
      {
        "agentId": "LMufO6dzRiSguShJOsJ53Q",
        "pid": "13992",
        "state": "AVAILABLE"
      }
    ]
  }
}
```

Additional queries

Additional OEJMX queries can be found in "OEJMX Query Reference" in *PAS for OpenEdge Administration Tools Reference*, including:

- Get agents query
- Get active session information query
- Get a stack trace for a session query
- Get metrics for an ABL application query
- Get hung client connections for an ABL application query
- Get agent metrics query
- Get agent session metrics query
- Get active threads from Tomcat thread pool query
- Refresh agent query
- Refresh all agents query
- Flush deferred log buffer to agent log file query
- Reset deferred log buffer query
- Get dynamic ABL session limit query
- Update dynamic ABL session limit
- Reset dynamic ABL session limit query

OEJMX utility

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

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

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

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

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

Syntax

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

Parameters

-C

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

-Q query_parameter

Specifies the path to a file with a set of JMX queries or a single JMX query. For more information about writing JMX queries, see Use OEJMX to manage and monitor an instance on page 44.

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

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

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

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

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

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

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

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

`-O output_file_name` `(with path)`

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

Default values:

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

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

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

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

`-R`

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

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

Automate lifecycle event scripts

Tomcat servers have the ability to start lifecycle events. These events include the startup and shutdown of a server. Administrators can write scripts to automatically run when these events occur. For example, scripts can start a third-party server when an instance starts. Logically, another script shuts down the third-party server when the instance shuts down. The scripts must be located in the *`instance-name`*`/bin` directory and use the following naming conventions.

Table 1: Automated lifecycle scripts

Script	Lifecycle event
*scriptPrefix*_startup.bat|sh	Executes when the Tomcat container is started but **before** any web application context are created.
*scriptPrefix*_started.bat|sh	Executes when the Tomcat container is started and **after** all web application contexts are created.
*scriptPrefix*_stopping.bat|sh	Executes **before** stopping an instance
*scriptPrefix*_shutdown.bat|sh	Executes **after** stopping an instance.

Note: Replace the *scriptPrefix* with a unique name for each instance.

Scripts execute inline

The startup and shutdown event scripts execute in line which means that the Tomcat startup/shutdown process waits for the event script to complete before continuing. If a script hangs, then it prevents the lifecycle event from completing.

Logs and errors

The standard server logs, located in the *instance-name*/logs directory, track when the scripts are called. If errors are raised by the script, then those errors do **not** interrupt the standard server startup or shutdown. Administrators are responsible for catching errors in their scripts and writing those errors to a script-specific log file in the *instance-name*/logs directory.

New content for management REST APIs

OpenEdge 11.7.4 adds the following documentation updates to support oemanager.war REST APIs added in 11.7.3:

- Check ABL object tracking status on page 54
- Enable or disable ABL object tracking on page 56
- Get a report of tracked ABL objects on page 57
- Terminate a specific ABL session on page 59
- Terminate Free ABL Sessions in a given agent on page 60

Check ABL object tracking status

Checks whether ABL object tracking is enabled or disabled. To change the status, see Enable or disable ABL object tracking on page 56.

HTTP Operation

GET

URI

```
http(s)://host_name:port/oemanager/applications/ABL_app_name/agents/agentPID/ABLObjects/status
```

Note: Agent PID (for example, `1144`) or agent ID (for example, `W9UI_6dJQ-qSsk-o25mWAw`) can be used to identify the multi-session agent. Both values can be obtained using the List agents API.

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -u username:password -v
http://localhost:16680/oemanager/applications/oepas1/agents/18668/ABLObjects/status
```

ABL HTTP client example

```
define variable resp as OpenEdge.Net.HTTP.IHttpResponse.

resp = OpenEdge.Net.HTTP.ClientBuilder:Build():Client
   :Execute(OpenEdge.Net.HTTP.RequestBuilder

:Get('http://localhost:16680/oemanager/applications/oepas1/agents/18668/ABLObjects/status')

      :UsingCredentials(new OpenEdge.Net.HTTP.Credentials('', 'username', 'password'))
      :Request
   ).
```

Request body example

NA

Response body example

Result is false when tracking is off. Result is true when tracking is on. In this example ABL object tracking is disabled.

```
{
   "operation":"TRACKING ABL OBJECTS",
   "outcome":"SUCCESS",
   "result":false,
   "errmsg":"",
   "versionStr":"v12.1.0 ( 2019-08-07 )",
   "versionNo":1
}
```

Enable or disable ABL object tracking

Description

Enable or disable the tracking of ABL objects. When enabled, ABL objects are tracked to identify potential memory leaks. To generate a report of these objects, see Get a report of tracked ABL objects on page 57. When object tracking is not required, disable. Disabled by default.

HTTP Operation

PUT

URI

http(s)://*host_name*:*port*/oemanager/applications/*ABL_app_name*/agents/*agentPID*/ABLObjects/status

Note: Agent PID (for example, 1144) or agent ID (for example, W9UI_6dJQ-qSsk-o25mWAw) can be used to identify the multi-session agent. Both values can be obtained using the List agents API.

Media type

application/vnd.progress+json

Response codes

200 Success
500 Unexpected Server Error

Command-line example

```
curl -X PUT -u username:password -v
http://localhost:16680/oemanager/applications/oepas1/agents/18668/ABLObjects/status
```

ABL HTTP client example

```
using Progress.Json.ObjectModel.*.
define variable resp as OpenEdge.Net.HTTP.IHttpResponse.
define variable jsonData as JsonObject.

jsonData = new Progress.Json.ObjectModel.JsonObject().
jsonData:Add('enable',  'true').
resp = OpenEdge.Net.HTTP.ClientBuilder:Build():Client
   :Execute(OpenEdge.Net.HTTP.RequestBuilder

:Put('http://localhost:16680/oemanager/applications/oepas1/agents/18668/ABLObjects/status,
 jsonData)
      :ContentType('application/vnd.progress+json')
      :UsingCredentials(new OpenEdge.Net.HTTP.Credentials('', 'username', 'password'))
      :Request
   ).
```

Request body example

```
{"enable":{"true" |  "false"} }
```

Response body example

Result returns true when value is changed. To get the current value, see Check ABL object tracking status on page 54.

```
{
   "operation":"TRACKING ABL OBJECTS",
   "outcome":"SUCCESS",
   "result":true,
   "errmsg":"",
   "versionStr":"v12.1.0 ( 2019-08-07 )",
   "versionNo":1
}
```

Get a report of tracked ABL objects

Description

Get a report of ABL objects that may indicate potential memory leaks. Getting a report requires that object tracking is enabled. See Enable or disable ABL object tracking on page 56. After leaks have been identified and resolved, disable ABL object tracking.

HTTP Operation

`GET`

URI

`http(s)://`*`host_name`*`:`*`port`*`/oemanager/applications/`*`ABL_app_name`*`/agents/`*`agentPID`*`/ABLObjects`

Note: Agent PID (for example, `1144`) or agent ID (for example, `W9UI_6dJQ-qSsk-o25mWAw`) can be used to identify the multi-session agent. Both values can be obtained using the List agents API.

Media type

`application/vnd.progress+json`

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X GET -u username:password -v
http://localhost:16680/oemanager/applications/oepas1/agents/18668/ABLObjects
```

ABL HTTP client example

```
define variable resp as OpenEdge.Net.HTTP.IHttpResponse.

resp = OpenEdge.Net.HTTP.ClientBuilder:Build():Client
   :Execute(OpenEdge.Net.HTTP.RequestBuilder

:Get('http://localhost:16680/oemanager/applications/oepas1/agents/18668/ABLObjects')
      :UsingCredentials(new OpenEdge.Net.HTTP.Credentials('', 'username', 'password'))

      :Request
   ).
```

Request body example

NA

Response body example

This excerpt shows the objects that are being tracked. `Source` identifies the related code.

```
{
    "result": {
        "ABLOutput": {
            "ABLObjects": [
                {
                    "AgentSessionId": 4,
                    "Objects": []
                },
                {
                    "AgentSessionId": 7,
                    "Objects": []
                }
            ]
        },
        "ABLReturnVal": true,
        "agentId": "y11SsN0cSC2skp9NV-N0Ww",
        "pid": "6064"
    },
    "errmsg": "",
    "versionStr": "v11.7.4 ( 2018-09-17 )",
    "versionNo": 1,
    "outcome": "SUCCESS",
    "operation": "GET ABL OBJECTS REPORT"
}
```

This excerpt shows results if object tracking is disabled.

```
{
    "result": {
```

```
        "ABLReturnVal": false,
        "agentId": "y11SsN0cSC2skp9NV-N0Ww",
        "pid": "6064"
    },
    "errmsg": "",
    "versionStr": "v11.7.4 ( 2018-09-17 )",
    "versionNo": 1,
    "outcome": "SUCCESS",
    "operation": "GET ABL OBJECTS REPORT"
}
```

Terminate a specific ABL session

Description

Terminate a specific ABL Session in a given agent.

HTTP Operation

DELETE

URI

//host_name:*port*/oemanager/applications/*App_name*/agents/*AgentID*/sessions/*ABLsessionID*

Media type

application/vnd.progress+json

Response codes

200 Success
500 Unexpected Server Error

Command-line example

```
curl -X DELETE -u username:password -v
http://localhost:16680/oemanager/applications/oepas1/agents/123/sessions/001
```

Request body example

NA

Response body example

```
{
  "result": true,
  "versionStr": "v11.7.3 ( 2018-04-27 )",
  "outcome": "SUCCESS",    "errmsg": "",
  "versionNo": 1,
  "operation": "TERMINATE ABL SESSION"
}
```

Terminate Free ABL Sessions in a given agent

Description

Terminate all free ABL Sessions in a given agent.

HTTP Operation

`DELETE`

URI

`//host_name:port/oemanager/applications/App_name/AgentID/sessions`

Media type

```
application/vnd.progress+json
```

Response codes

```
200 Success
500 Unexpected Server Error
```

Command-line example

```
curl -X DELETE -u username:password -v
http://localhost:16680/oemanager/applications/oepas1/GjfLygRURtSnD73H7dnJ0g/sessions
```

Request body example

NA

Response body example

```
{
    "result": 2,
    "operation": "TERMINATE SESSIONS",
    "versionStr": "v11.7.3 ( 2018-04-27 )",
    "errmsg": "",
    "versionNo": 1,
    "outcome": "SUCCESS"
}
```

Secure an instance

Secure an instance with these enhancements to PAS for OpenEdge.

- Authenticate using SAML on page 61
- About support for server name indication on page 69

Authenticate using SAML

Security Assertion Markup Language (SAML) is an XML-based specification that defines how an identity provider (a SAML authority) and a service provider can exchange authentication and authorization data. It is developed and maintained by the Organization for the Advancement of Structured Information Standards (OASIS).

SAML is a network protocol. It stipulates that an identity provider must generate a SAML assertion containing information about an authenticated user, that is then transported over HTTP to a SAML service provider. A single SAML assertion can be used as an authentication token by many service providers across different security domains, enabling enterprises to implement cross-domain single sign-on (SSO).

SAML-based SSO benefits both application users and enterprises. It enables users to access multiple applications without having to log in each time. For enterprises, using an identity provider as a SAML authority gives them the ability to manage users, roles, and permissions from a central authentication and authorization repository.

PAS for OpenEdge supports SAML 2.0. A PAS for OpenEdge web application can be configured to act as a SAML service provider, thus enabling an OpenEdge domain to participate in a cross-domain SAML SSO implementation.

Note that PAS for OpenEdge does not act as a direct-login service provider. Instead, it is the responsibility of a PAS for OpenEdge client application to accept requests from an end user, and communicate with an identity provider to obtain SAML assertions for authenticated users.

After the PAS for OpenEdge client application receives a SAML assertion from an identity provider, it sends the assertion to a PAS for OpenEdge web application, that then validates the assertion. If the validation is successful, then PAS for OpenEdge automatically creates an ABL CLIENT-PRINCIPAL object, that is used to authenticate the user across all parts of the ABL application, including its databases, thus implementing SSO all the way through the application.

About SAML

The growth in the adoption of SAML is closely linked to the proliferation of SaaS applications. As users begin to access the bulk of their applications over the web, enterprises need to ensure that user identities are centrally managed and delivered to application providers in a secure, standard way. The SAML specification addresses this need by identifying the roles of the parties involved in exchanging authentication data and specifying a standard way that authentication data is structured and delivered.

The current version of the specification is **SAML 2.0**. The concepts described in this topic are a small subset of the SAML specification; they relate to only those elements that you need to understand to implement SAML in PAS for OpenEdge.

For more information about SAML, see the official SAML technical overview.

Participants

In a typical SAML-based SSO workflow, there are three key participants:

- A **user**, also called a **principal**, who needs to access a service and whose identity needs to be authenticated.
- A **SAML service provider**, which is typically a web application that provides services to authenticated users and can process SAML assertions.
- An **identity provider**, also known as a **SAML authority**, that manages user identities and provides authentication and authorization services. Examples of identity providers include Okta, OneLogin, and Active Directory Federation Services.

In addition, a user may require a user agent, such as a browser, to access the desired services.

Workflow

At a high-level, this is a SAML-based SSO workflow:

1. A user tries to access a resource or service hosted by the service provider.
2. The service provider creates a SAML request and redirects the user's browser to a configured SSO URL that belongs to an identity provider.
3. The identity provider processes the SAML request and presents an authentication challenge such as a login page or a two-factor/multi-factor authentication form to the user.
4. The user responds to the authentication challenge.
5. If the identity provider is able to authenticate the user, then it sends a SAML response containing a SAML assertion to the user's browser and redirects the browser to an assertion consumer service (ACS) URL that belongs to the service provider.
6. The service provider receives the SAML response and validates the SAML assertion. It then grants access to the requested service or resource.

This type of workflow is called Service Provider Initiated SSO. There is also another type of workflow that is less common, called Identity Provider Initiated SSO, where a user logs in to the identity provider first and then accesses a service provider page.

Protocols, bindings, and profiles

The SAML specification caters to a wide range of authentication requirements. As a result, it defines a number of protocols, bindings, and profiles, each of which pertain to specific authentication use cases.

A *protocol* is a standard way to perform an operation. For example, the SAML specification defines an authentication request protocol that service providers must use when they need to send authentication requests to an identity provider.

A *binding* defines how a protocol message must be carried over a transport layer. The SAML specification defines bindings for various types of HTTP and SOAP requests.

A *profile* is a combination of bindings and protocols. For example, the Web Browser SSO Profile defines that when a service provider needs to authenticate a user who is trying to access a resource using a browser, it should send an authentication request message to an identity provider using either the HTTP Redirect, HTTP POST, or HTTP Artifact binding.

SAML assertions

At the core of SAML is a SAML assertion (a SAML token). When an identity provider authenticates a user's identity, it generates a SAML assertion that it returns as part of a SAML response. A SAML assertion in the Web Browser SSO Profile contains the following:

- Information about the assertion itself, such as the SAML version, the timestamp when the assertion was issued, the identity provider that issued it, and so on.
- Information about the user (referred to in the assertion as the **subject**). The assertion typically specifies the Name ID of the user (joe@hiscompany.com) and also specifies the format of the Name ID (email address, Windows domain qualified name, Kerberos principal name, and so on).
- Validity period of the assertion; may include conditions such as not before or after a specified date and time.
- An `<AudienceRestriction>` element that specifies which URIs are valid for the assertion. The Assertion Consumer Service (ACS) URL must be listed as one of the audiences in this element.
- Optional attributes that can be customized based on your needs. A typical use of these optional attributes is to pass information about the user's roles.

A SAML response can be signed or unsigned, and a SAML assertion can be signed or encrypted. The service provider and the identity provider must be configured so that there is an 'agreement' on the signatures and encryptions. This is usually done by exposing configurations through metadata.

Metadata

Because digital signatures and encryption can be used in SAML messages, the service provider and the identity provider must establish trust in advance. This is typically done through metadata defined in an XML file.

The metadata of an identity provider includes information such as:

- The entity ID, which is a unique identifier for the identity provider. This is typically an endpoint or URL.
- Supported HTTP bindings (such as POST or Redirect) and the SSO URL for each binding.
- The identity provider's X.509 public key certificate that is used by the service provider to decrypt SAML response messages.

The metadata of the service provider includes information such as:

- The entity ID, which is a unique identifier for the service provider. This is typically an endpoint or URL.
- The Assertion Consumer Service (ACS) URL. This is the location where the service provider receives SAML response messages. The ACS URL is also sent in the SAML request message.
- The service provider's X.509 public key certificate that is used by the identity provider to decrypt SAML request messages.

There are a few elements whose values must match in the metadata of the identity provider and the service provider, for example:

- The name ID format of the user whose identity needs to be authenticated.
- The hashing algorithm if digital signatures are being used.

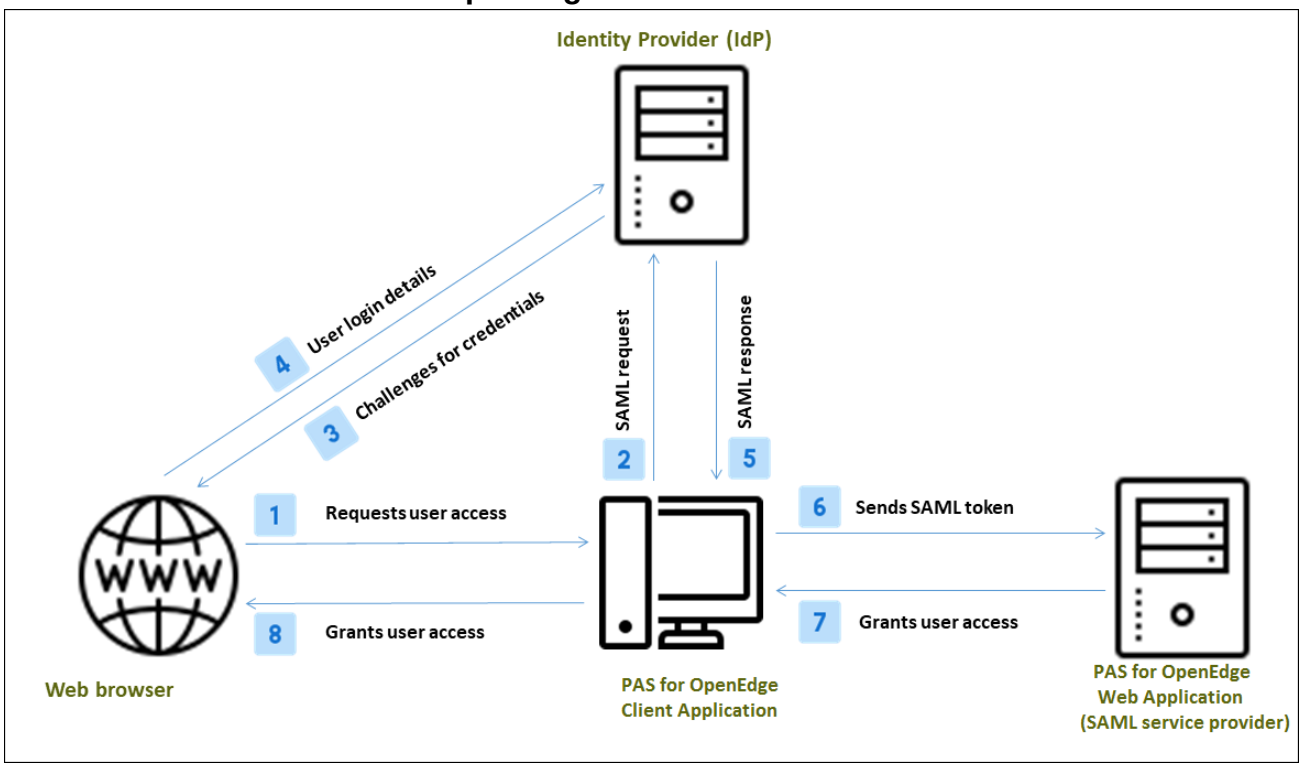
About support for SAML in PAS for OpenEdge

It is important to note that PAS for OpenEdge does not serve as a direct-login service provider. This means that end users do not directly access a PAS for OpenEdge web application. Instead, a PAS for OpenEdge client application acts as the intermediary between the user and the PAS for OpenEdge web application. A PAS for OpenEdge client application can be any REST or JavaScript program that can send and receive HTTP messages.

The client application is also responsible for sending SAML requests to an identity provider, receiving a SAML assertion from it, and then sending the assertion to the PAS for OpenEdge web application. The assertion must be sent using the basic authentication header in an HTTP request.

To set up a PAS for OpenEdge web application as a SAML service provider, you must configure the web application's `oeablSecurity.properties` file.

How SAML works in PAS for OpenEdge



A SAML workflow that involves PAS for OpenEdge is similar to any service provider initiated SSO workflow, but with a few additional steps:

1. A user connects to a PAS for OpenEdge client application using a web browser and tries to perform an operation that requires a resource deployed on a PAS for OpenEdge web application.
2. The PAS for OpenEdge client application generates a SAML authentication request and redirects the user's browser to the configured identity provider's SSO URL.
3. The identity provider processes the SAML request and presents the user with an authentication challenge.
4. The user responds to the authentication challenge by supplying the required credentials.
5. The identity provider authenticates the user and generates a SAML response containing a SAML assertion and redirects the user's browser to the client application's ACS URL.
6. The PAS for OpenEdge client application receives the SAML response, processes it, and sends the SAML assertion to the PAS for OpenEdge web application by passing it in an HTTP Authorization header.
7. The PAS for OpenEdge web application processes the SAML assertion based on properties configured in its `oeablSecurity.properties` file. It performs tasks such as:
   a. **Decryption**: If the assertion is encrypted or signed, the PAS for OpenEdge web application uses the identity provider's public key (exposed using the provider's metadata) to decrypt the message.
   b. **Validity Period Check**: The PAS for OpenEdge web application verifies that the assertion is not past its expiration date.
   c. **First Level HTTP Method Validation**: The PAS for OpenEdge web application checks if the HTTP method used by the PAS client to send the SAML assertion is a permitted method.
   d. **(Optional) ACS URL Validation**: Depending on the property settings, the PAS for OpenEdge web application may also match the assertion's <`AudienceRestriction`> element against the PAS client's ACS URL (exposed using the client's metadata).
   e. **Authorization**: The PAS for OpenEdge web application obtains the user's roles from attribute statements in the SAML assertion and determines if the user is authorized to access the requested resource. It does this by matching the roles obtained from the assertion with the roles defined for resource URLs in the `oeablSecurity.csv` file.
   f. **Second level HTTP method validation**: The PAS for OpenEdge web application verifies that the HTTP method used in the request matches the permitted HTTP method defined for the requested resource URL in the `oeablSecurity.csv` file.
   g. **CLIENT-PRINCIPAL Creation**: If the PAS for OpenEdge web application is able to perform the previous tasks successfully, it creates a sealed CLIENT-PRINCIPAL object and passes the object to the relevant ABL resource.
   h. **Response**: Finally, the PAS for OpenEdge web application sends a response to the client application.
8. The client application processes the response and grants access to the end user, to perform the operation.

How SAML assertion elements map to CLIENT-PRINCIPAL object attributes

If the PAS for OpenEdge web application is able to validate the SAML assertion, it creates a sealed CLIENT-PRINCIPAL object and passes it to the required ABL resource.

The following attributes of the resulting CLIENT-PRINCIPAL object are populated from the SAML assertion's fields:

CLIENT-PRINCIPAL attribute	SAML assertion field
USER-ID[1]	NameID[1]
DOMAIN-NAME[1]	NameID[1]
SESSION-ID	SessionIndex attribute in AuthnStatement
LOGIN-EXPIRATION-TIMESTAMP[3]	NotOnOrAfter attribute in Conditions
ROLES[2]	Optional attributes for roles [2]

[1] If the NameID is an email address, PAS for OpenEdge uses the at sign ('@') separator to extract the user ID and the domain name.

[2] The names of these optional attributes must be set in the samlToken.UserDetails.roleAttrName property in the oeablSecurity.properties file.

[3] The oeablSecurity.properties file has a property named samlToken.webSSOProcessingFilter.responseSkew. If a value is specified for this property (in seconds), it is added to the NotOnOrAfter timestamp.

Other attribute values are populated based on the OEClientPrincipalFilter properties set in the oeablSecurity.properties file.

If there are other attributes that you want to set, that are not covered by either the SAML assertion or the default OEClientPrincipalFilter, then you can set them using any of the 10 key-value OEClientPrincipalFilter static properties. For example, if you want to set the DOMAIN-DESCRIPTION attribute, then you can modify a key-value property:

```
OEClientPrincipalFilter.properties.1.key=DOMAIN-DESCRIPTION
 OEClientPrincipalFilter.properties.1.value=OEApplication
```

Implement SAML in a PAS for OpenEdge web application

To implement SAML in PAS for OpenEdge, you must perform two tasks:

- Set up a PAS for OpenEdge client application.
- Configure SAML properties in a PAS for OpenEdge web application.

Set up a PAS for OpenEdge client application

Because a PAS for OpenEdge web application does not act as a direct-login service provider, you need to develop a PAS for OpenEdge client application that can communicate with end users and work with identity providers.

The common use case is to develop a client application (for example, a Kendo UI Builder application) that provides HTML pages that end users access using a browser. When the end user clicks a link or URL that requires a protected OpenEdge resource, the PAS for OpenEdge client initiates a SAML workflow. This resource could be, for example, an ABL business entity wrapped in a Data Object Service, in which case you would want your PAS client to call a URL such as http://*host*:*port*/*OpenEdgeApplication*/rest/*DataObjService*/*ABLBusinessEntity*.

To implement this use case, you must develop your PAS client application and set it up to work with an identity provider of your choice. The important thing to remember is that the PAS client application must send the SAML assertion that it receives from an identity provider to the PAS for OpenEdge web application.

Note:

To obtain a protected resource from PAS for OpenEdge, the client application must send a valid SAML assertion to the PAS for OpenEdge web application. This assertion *must* be sent in the **Authorization header** of the HTTP request with the prefix **SAML**.

Also, the HTTP method that your PAS client uses to send SAML assertions must match the `samlToken.httpBinding.allowedMethods` property set in the `oeablSecurity.properties` file. It must also match a method defined for the resource URL in the `oeablSecurity.csv` file.

In addition, you need to generate a metadata file for your PAS client application. This task is performed either by an application developer or a system administrator. There are tools available online (such as this one) that can help generate this metadata. The metadata is useful not only when you are configuring the identity provider to communicate with your PAS for OpenEdge client application, but also when you are configuring SAML properties in the PAS for OpenEdge web application.

Configure SAML properties in a PAS for OpenEdge web application

After you set up your client application, configure the following properties in your PAS for OpenEdge web application's `oeablSecurity.properties` file located in the `WEB-INF` folder.

Note: Before you configure these properties, download the metadata file of the client application and the identity provider, and store them in a location that is accessible to your PAS for OpenEdge web application.

Client login model

Set the value of the `client.login.model` property to `saml`:

```
client.login.model=saml
```

Metadata

Set the path to the PAS client application's (service provider's) metadata.

```
samlToken.metadata.spMetaDataFileLocation=<path to PAS client application's metadata>
```

Set the path to the identity provider's metadata.

```
samlToken.metadata.idpMetaDataFileLocation=<path to IdP's metadata>
```

HTTP method validation

Set the allowed HTTP methods for incoming HTTP requests that contain the SAML assertion. If the only allowed method is `GET`, and the SAML assertion is part of a `POST` message, PAS for OpenEdge rejects the request.

```
samlToken.httpBinding.allowMethods=GET,POST,PUT,DELETE
```

ACS URL validation

The `Audience restriction` URL, a field in the SAML assertion that Identity Provider returns, must match the location value of the `AssertionConsumerService` URL in the Service Provider's metadata file. If not, the Service Provider rejects the SAML response from the Identity Provider. This property is optional and the URL matching can be skipped. However, if you need to check ACS endpoint, set this property to `true`.

```
samlToken.webSSOProcessingFilter.checkForACSEndpointUrl={false|true}
```

Validity period

PAS for OpenEdge rejects a request if the SAML assertion's `NotOnOrAfter` timestamp has passed. However, you can set the following property to extend the duration of the assertion's validity period.

```
samlToken.webSSOProcessingFilter.responseSkew=<time in seconds>
```

Roles

If the incoming SAML assertion's attribute statements define user roles, you can set the attribute names as comma-separated values in the following property. This enables PAS for OpenEdge to populate the `ROLES` attribute when creating the CLIENT-PRINCIPAL object.

```
samlToken.UserDetails.roleAttrName=Attribute1,Attribute2
```

Note that PAS for OpenEdge authorizes a user to access a resource only if the role values in the SAML assertion's attributes map to roles defined for resource URLs in the `oeablSecurity.csv` file. By default, these roles have a `ROLE_` prefix. If the incoming role values do not contain the prefix, you an use the following property to add it:

```
samlToken.UserDetails.rolePrefix=ROLE_
```

Domains

If the name ID passed in the SAML assertion is an email address (for example *johndoe@hisdomain.com*), PAS for OpenEdge extracts the domain name using the '@' separator and assigns it to the `DOMAIN-NAME` attribute of the CLIENT-PRINCIPAL object that it creates. However, if the name ID is not an email address, you can use the following property to set a default domain name:

```
samlToken.UserDetails.userDomain=Domain_Name
```

You may want to specify a domain name that has minimal access rights to your ABL application.

Note: If `saml` is set as the client login model, this property overrides the `OEClientPrincipalFilter.domain` property.

To create a sealed CLIENT-PRINCIPAL object, PAS for OpenEdge requires a domain access code that corresponds to the domain name. Typically, a CSV file is created containing a list of key-value pairs (for example, *domain, domain-access-code*) and then encrypted into a Java keystore file. To specify this keystore file, use the following property:

```
samlToken.UserDetails.registryFile=keystore_file
```

Note: If `saml` is set as the client login model, this property overrides the `OEClientPrincipalFilter.registryFile` property. For more information about creating registry files, see Configuring multiple domains.

Debug logging for SAML

Due to various role implementations by different vendors, you may have to troubleshoot your SAML configuration when a problem occurs. You can enable OpenEdge logging and use the required information to debug the problem.

Open the *web-app*/WEB-INF/logging.xml file. Un-comment the following lines, and set their values to "DEBUG":

```
<logger name="org.springframework.security.saml" level="DEBUG"/>
<logger name="org.opensaml" level="DEBUG"/>
<logger name=" PROTOCOL_MESSAGE " level="DEBUG"/>
```

About support for server name indication

If your OpenEdge Authentication Gateway server is configured with multiple virtual hosts that are bound to a single IP address, then you can use a parameter called -servername to specify the hostname that you want to connect to in the STS Client and the STS URL utilities.

STS client utility (stsclientutil)

The stsclientutil utility is an administrator-level utility used in troubleshooting OpenEdge database connections to the Secure Token Server (STS) application. Specifically, stsclientutil addresses these tasks:

- Establishes the OpenEdge installation's ability to connect to a PAS for OpenEdge server and use its STS application (that is, *ping* the STS).
- Tests the ability of the OpenEdge installation to use an STS application for a direct user login operation for a specific OpenEdge domain (that is, *authenticate* the user).
- Tests the ability of the OpenEdge installation to use an STS application for an operating-system, user login SSO to a specific OpenEdge domain (that is, *exchange* the OS security token for an OpenEdge security token).

Command syntax

```
stsclientutil -cmd command -url sts-url
      [command-options]
```

-cmd *command*

Specifies the task to perform from one of the following:

- **ping**—Sends an empty message to an STS application to test HTTPS connections and STS key features.
- **authenticate**—Sends user login credentials to the STS application to test its domain configuration.

With `authenticate`, you must also specify the OpenEdge user ID used to send to the STS application for direct user login testing:

```
-user user[@domain]
```

- **exchange**—Sends the security token of the current process's operating system user login to the STS application to test its domain configuration.

  With `exchange`, you must also specify the OpenEdge domain name to send to the STS application for operating system SSO testing:

  ```
  -domain domain
  ```

Note: There is no default for *command*; one of the options must be specified.

`-url` *sts-url*

Specifies the URL of the STS application (secure PAS for OpenEdge server that includes a security token service (STS) application) used by the OpenEdge database. There is no default for *sts-url*; the URL must be specified.

command-options

The following options are available for each of the `-cmd` choices:

`-sslversion { TLSv1 | SSLv2 | SSLv3 | TLSv1.0 | TLSv1.1 | TLSv1.2 | TLSv1.3 }`

Specifies TLS protocol to use when connecting to the PAS for OpenEdge server hosting the STS application. If not specified, the default is `TLSv1.2`.

`-sslciphers` *cipher*`[,`*cipher*`...]`

Specifies one or a list of TLS cipher suites to use when connecting to the PAS for OpenEdge server hosting the STS application. If not specified, the default is `TLSv1.2` for all ciphers.

`-installpath` *dlc-path*

Specifies the operating system file path of the OpenEdge installation to test, in the event of multiple OpenEdge installations. If not specified, the default is `$DLC`or `%DLC%`.

`-keystorepath` *ks-path*

Specifies the operating system file path to where the STS client key is installed. If not specified, the default is `$DLC/keys`.

The STS client key is installed by a DBA who has physical access to the OpenEdge Authentication Gateway server key. The key is managed with the `stskeyutil` utility.

`-certstorepath` *`cs-path`*

Specifies the operating system file path to where the HTTPS (TLS) CA certificates can be found to validate the PAS for OpenEdge server's certificate. If not specified, the default is `$DLC/certs`.

`-logginglevel` *`level`*

Specifies the amount of `stdout` logging to be provided during troubleshooting activities. If not specified, the default level is 2. The range is 0-5.

`-nohostverify`

Suppresses the validation of the PAS for OpenEdge certificate's subject name against the URL host DNS name.

The host name checking of TLS server certificates compares the name returned by a DNS lookup of the URL's *host* field to the *CN* subfield of the server certificate's *subject-name* X509 name.

`-servername`

Use this parameter when the OpenEdge Authentication Gateway server is configured with multiple virtual hosts that are bound to a single IP address. Specify the hostname that you want the STS client utility to connect to. The STS client utility requests the virtual host's TLS certificate during the TLS handshake instead of the Server URL's host.

The `stsclientutil` utility is a secured application that can be included into DBA automation scripts without fear of disclosing sensitive user credentials or OpenEdge security tokens that could be used to gain access to OpenEdge databases.

Examples

Examples of `stsclientutil` with the `ping` command:

```
stsclientutil -cmd ping -url https://sts.acme.com:8992

stsclientutil -cmd ping -url https://sts.acme.com:8992/oests
```

Examples of `stsclientutil` with the `authentication` command:

```
stsclientutil -cmd authentication -url https://sts.acme.com:8992 -user fred -nohostverify

stsclientutil -cmd authentication -url https://sts.acme.com:8992 -user 'fred@acme.admins'

stsclientutil -cmd authentication -url https://sts.acme.com:8992 -user 'fred@acme.admins'
 -sslversion TLSv1.0
```

Examples of `stsclientutil` with the `exchange` command:

```
stsclientutil -cmd exchange -url https://sts.acme.com:8992 -domain 'acme.osusers'

stsclientutil -cmd exchange -url https://sts.acme.com:8992 -domain 'acme.osusers'
-logginglevel 5
```

stsurlutil update utility

Change the OpenEdge Authentication Gateway server URL for each specified database.

Note: The change takes effect the next time the database is started.

```
stsurlutil update -url url

    [ -servername virtual-hostname]
    [ -folder folder-name]
    [-ssl args]
    [-test true | false]
    -db dbname [db-options]
         [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-url url`

Specifies the new URL for the Authentication Gateway to insert into the database configuration.

Specify the *url* using the format: `https://<host>[port][/sts-application-name]`

`-servername virtual-hostname`

Use this optional parameter when the Authentication Gateway is configured with multiple virtual hosts that are bound to a single IP address. Specify the hostname that you want the database to connect to. This creates a row in the database's options table (`_db-option`) that stores the virtual hostname. At runtime, when the database needs to connect to the Authentication Gateway, it requests the virtual host's TLS certificate during the TLS handshake instead of the server URL's host.

`-folder folder-name`

Specifies a directory (folder) where log files and temporary files are written. If not provided, the following locations are tested, and the first available folder is used:

- The directory specified by the `WRKDIR` environment variable
- Current working directory
- The session temp-dir specified using `-T`

`-ssl args`

Additional TLS (https) connection options. (See the ABL `socket:connect` method.)

`-test true | false`

Tests if the URL is reachable (connect-only.) The default is `true`, if not specified. Only the literal values `true` and `false` are accepted (no other traditional Boolean pairs).

`-db dbname [ db-options ]`

Specifies one or more OpenEdge databases to connect to. *db-options* allows you to specify additional connection parameters.

`[ -db dbname [ db-options ] ... ]`

Specifies additional databases and connection options.

abl-startup-options

Specifies additional ABL connection parameters.

The `stskeyutil update` utility writes the new URL of the gateway to the database.

Secure online deployment of a new ABL application

OpenEdge system administrators can securely deploy new ABL applications to production PAS for OpenEdge instances without having users experience any downtime. A secure ABL application deployment to a production instance requires a securely deployed Tomcat Manager, as well as other security considerations, that are described in this topic.

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

To enable secure the online deplomen of new ABL applications:

1. **Deploy Tomcat Manager.**

   To deploy ABL applications online, you need to use Tomcat Manger features. For security reasons, Tomcat Manager is not deployed, by default, on production servers.

   To deploy Tomcat Manager:

   ```
   proenv> pasman.[sh|bat] deploy -I instance-name $DLC/extras/manager.war
   ```

2. **Secure the Tomcat Manager remote address valve.**

   To maintain the security of your PAS for OpenEdge instance, do not enable remote administration on a production server. When you enable Tomcat Manager to allow online deployment of ABL applications, limit access to Tomcat Manager to localhost requests. Specify the localhost requests using the remote address valve configuration in *instance-name*`/webapps/manager/META-INF/context.xml`, as shown:

   ```
   <!--
   Remove the comment markers from around the Valve below to limit access to
   the manager application to clients connecting from localhost
   -->
   <!--
   <Valve className="org.apache.catalina.valves.RemoteAddrValve"
   allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
   -->
   ```

   The remote address valve compares the IP address of the client that submitted the request against one or more *regular expressions*, and either allows the request to continue or refuses to process the request from the client.

Note: With IPv6 addresses, the format of the IP address this valve processes depends on the API that was used to obtain it. If the address was obtained from a Java socket using the `Inet6Address` class, then its format is `x:x:x:x:x:x:x:x`. That is, the IP address for the localhost is `0:0:0:0:0:0:0:1` instead of the more widely used `::1`. Consult your access logs for the actual value.

3. **Employ additional security recommendations.**

   Using Tomcat Manager, or other management applications, requires you to carefully secure your environment. Because the Tomcat Manager application allows for web applications to be deployed remotely, it can be targeted by attackers. This is often due to publicly accessible Tomcat instances that have weak passwords and have the Manager application enabled. If the Manager application is enabled, then follow these security guidelines:

   - Ensure that any users allowed to access the Tomcat Manager application have strong passwords.
   - Change the default username and password.
   - Use roles to limit deployment capabilities to authorized users.
   - Use `LockOutRealm` (enabled by default) that prevents brute force attacks against user passwords. Do not disable `LockOutRealm`.
   - Use file system security to protect the web application configuration, and use the PAS for OpenEdge security configuration files.

4. **Deploy the ABL web application.**

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

   To deploy an `oeabl.war` based web application:

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

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

Troubleshoot problems

Trouble shoot problems with these enhancements to PAS for OpenEdge.

- Centralized logging in PAS for OpenEdge on page 75
- Enhanced Mapped Diagnostic Content logging in PAS for OpenEdge on page 82

Centralized logging in PAS for OpenEdge

As PAS for OpenEdge runs, it writes information to log files stored on local disks. You may want to centralize these log messages over the cloud using Elasticsearch, Filebeat, and Kibana, for the following reasons:

- By centralizing log messages over the cloud, you are creating a backup of your logs that you can use if a problem occurs and your local logs are lost (for example, if your PAS for OpenEdge instance stalls, or if you have applied a software patch without backing up your log files first).
- Large scale OpenEdge deployments include many other parts of the IT infrastructure, such as routers, firewalls, storage arrays, internal and external applications, etc. When a problem occurs, sorting through multiple differently-formatted log files that are stored in different locations can be a challenge. Instead, by centralizing all logs using Elasticsearch, Filebeat, and Kibana, you can view them easily over a browser in a standardized format.
- You can leverage Elasticsearch and Kibana's features to perform various types of analytical operations on the log messages. This is especially useful when you are trying to improve the performance of your PAS for OpenEdge instances or when you are troubleshooting an issue.

Elasticsearch, Filebeat, and Kibana

Elasticsearch, Filebeat, and Kibana are part of the Elastic Stack of products. Elasticsearch provides data storage capabilities. To use Elasticsearch as the central repository for your PAS for OpenEdge logs, you need Filebeat. Filebeat must be installed on every host machine that is running a PAS for OpenEdge instance that you want to monitor. Once installed, configured, and started, Filebeat acts as an agent process that reads your PAS for OpenEdge log files and sends log messages from those files to an Elasticsearch server. In addition, you can use Kibana to view those logs from a browser and perform various types of analytical operations.

Note: Centralized logging using Elasticsearch, Filebeat, and Kibana is not supported on AIX. Visit https://www.elastic.co/support/matrix to learn more about supported platforms.

How to use

Perform the following steps to centralize your PAS for OpenEdge logs with Elasticsearch:

1. Set up Elasticsearch.
2. Set up Filebeat.
3. Set up Kibana.

Set up Elasticsearch

Install and start Elasticsearch

You can either install Elasticsearch on-premise or use the Elasticsearch Cloud Service. If you are installing it on-premise, you need to have Java 8 on the host machine before you begin installation.

Download the Elasticsearch installation package for your operating system from the Elasticsearch downloads page.

Then, install and start Elasticsearch by following the instructions for your operating system as described in the Elasticsearch installation documentation.

Verify that Elasticsearch is running

After you start Elasticsearch, you can verify that it is running by sending an HTTP request to `localhost:9200` on the host machine. You should see a response that looks like this:

```
{
  "name" : "fgzodeC",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "t0gobiyRQ2Wh_h4gWOCGGg",
  "version" : {
    "number" : "6.4.0",
    "build_flavor" : "default",
    "build_type" : "zip",
    "build_hash" : "595516e",
    "build_date" : "2018-08-17T23:18:47.308994Z",
    "build_snapshot" : false,
    "lucene_version" : "7.4.0",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

For more information on configuring Elasticsearch, see the Configuring Elasticsearch documentation.

Set up Filebeat

How Filebeat works

The role of Filebeat, in the context of PAS for OpenEdge, is to send log messages to Elasticsearch. As part of setting up Filebeat, you must minimally configure two properties--the **filepaths of your log files** and the **connection details of Elasticsearch**.

Filebeat has two key components: **inputs** and **harvesters**. The inputs component uses the filepaths that you configure to find files that need to be read. For each file, it starts a harvester. Each harvester opens its assigned file, reads it line by line, and sends the contents to Elasticsearch. It continues to do this until configuration settings in Filebeat instruct it to close.

If a log file is deleted or moved, as may happen if you use TCMAN `clean` or if your PAS for OpenEdge instance's log files roll over, the harvester may continue to try to read the file, reserving space on your disk until it closes. To address situations like this, you need to correctly configure Filebeat.

To learn more about Filebeat, see How Filebeat works.

Install Filebeat

You must install Filebeat on every host machine that is running a PAS for OpenEdge instance that you want to monitor.

Download Filebeat for your operating system from the Download Filebeat page.

Then, install it by following the instructions for your operating system in the Install Filebeat documentation.

Configure Filebeat

To configure Filebeat, edit the `Filebeat.yml` configuration file that is located in the Filebeat installation directory. Minimally, you need to configure two properties:

- The filepaths of the directories that contain the PAS for OpenEdge logs that you want to centralize. You may typically want to include the Tomcat server logs (catalina, localhost, localhost_access), ABL application logs, and PAS for OpenEdge agent logs.
- Connection details of the Elasticsearch server. This includes the host, port, username, and password.

The following is an example that shows how to configure the filepaths and the connection details:

```
filebeat.inputs:
  - type: log
     enabled: true
     paths:
      - /PASInstance1/logs/*.log
      - /PASInstance2/logs/*.log

output.elasticsearch:
   hosts: ["192.168.1.42:9200"]
   username: "admin"
   password: "adminpassword"
```

Inputs and input types

The `filebeat.inputs:` section enables you to configure the input component of Filebeat. You can configure different input types (using the `- type` key as in the example), including `log`, `Stdin`, etc.

You can specify multiple inputs, and you can specify the same input type more than once. You can specify individual log filenames or specify `*.log`, which will instruct Filebeat to read all log files in the directory.

```
filebeat.inputs:
  - type: log
     enabled: true
     paths:
      - /PASInstance1/logs/*.log
      - /PASInstance2/logs/*.log
  - type: log
     enabled: true
     paths:
      - /PASInstance3/logs/oepas1.agent.log
```

Other input configuration options depend on the input type. The following are some key configuration options for the log input type.

Filtering logs

The following configuration options enable you to filter log messages before they are sent to Elasticsearch. These options must be specified at the same indentation level as the `paths:` option.

`exclude_lines`

A list of regular expressions that matches the lines that you want to exclude. For example, `exclude_lines: ['^INFO']`.

`include_lines`

A list of regular expressions that matches the lines that you want to include. Only lines that pass this filter will be sent to Elasticsearch. For example, `include_lines: ['^ERR', '^WARN']`.

`exclude_files`

A list of regular expressions that matches the filenames that you want Filebeat to ignore.

File handling

The following configuration options determine how the Filebeat harvesters handle files.

`scan_frequency`

A time period that determines how often Filebeat checks for new files in the log directory. This is applicable only if you have specified `*.log` for a path in the `paths:` configuration. Use the syntax <*Time*>s (for example, `10s` for 10 seconds).

`close_inactive`

A time period that determines a period of inactivity after which Filebeat closes the file. The time period begins when the last line in the log file is read by the harvester. For example, if you specify `3s` (3 seconds), and no new log messages are written to the file in the 3 seconds since the last line was read, Filebeat closes the file. Use the syntax <*Time*>s (to specify time in seconds), <*Time*>m for minutes, or <*Time*>h for hours.

`close_removed`

A Boolean value of `true` or `false`. If set to `true`, which is the default, Filebeat closes the file if it has been moved or deleted.

To read about other configuration options for the log input type, see Filebeat input log parameters.

Start Filebeat

To set up Filebeat as a Windows service, run the PowerShell script `install-service-filebeat` located in the Filebeat installation directory. Then start Filebeat either from `services.msc` or by entering `Start-Service filebeat` in a command prompt that points to the Filebeat installation directory. To start Filebeat in the foreground in a Windows operating system, open a command prompt, change the directory to the Filebeat installation folder, and then enter `filebeat.exe -e`.

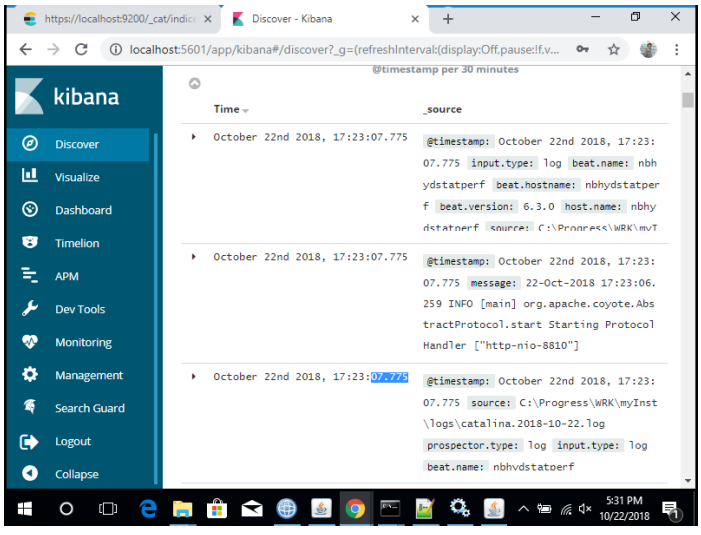
If you are using other operating systems, see the Starting Filebeat documentation.

This starts a shipper or daemon process that reads from the log files and sends the log entries to Elasticsearch.

Note: Before you start Filebeat, ensure that your Elasticsearch server is up and running.

Set up Kibana

Kibana is an Elasticsearch product that enables you to visualize your Elasticsearch data through a browser. You can use it to view and analyze your PAS for OpenEdge log messages that are stored in Elasticsearch.



Install Kibana

You can either install Kibana on-premise or, if you are using the Elasticsearch Cloud Service, access it directly by logging in through a browser.

If you are installing Kibana on-premise, download the Kibana installation package for your operating system from the Kibana downloads page. Then, install Kibana by following the instructions for your operating system as described in the Kibana installation documentation.

Configure Kibana

The Kibana installation directory contains a file named `kibana.yml` in the `config` folder. Edit this file in a text editor and set the following properties to enable Kibana to connect to your Elasticsearch server:

```
elasticsearch.url: "http://<server_hostname>:<server_port>"
elasticsearch.username: "<username>"
elasticsearch.password: "<password>"
```

Start Kibana

After you have configured Kibana, start it by following the instructions for your operating system as described in the Starting and stopping Kibana documentation.

For example, in Windows, run `./bin/kibana.bat` in a command line console.

Launch the Kibana web application

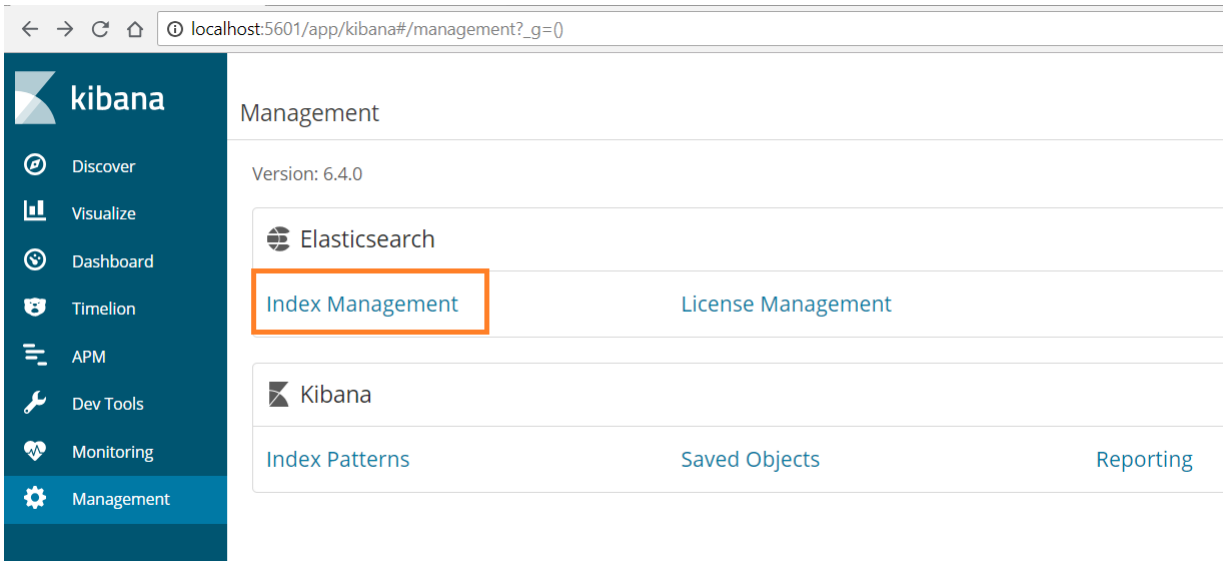
Kibana provides a web application to enable you to view your PAS for OpenEdge logs in a browser. Once Kibana is up and running, you can launch the web application by entering its URL `http://<hostname>:<port>` in a browser (`http://localhost:5601` by default).

Note: To change the default port, modify the `server.port` setting in the `kibana.yml` file and restart Kibana.

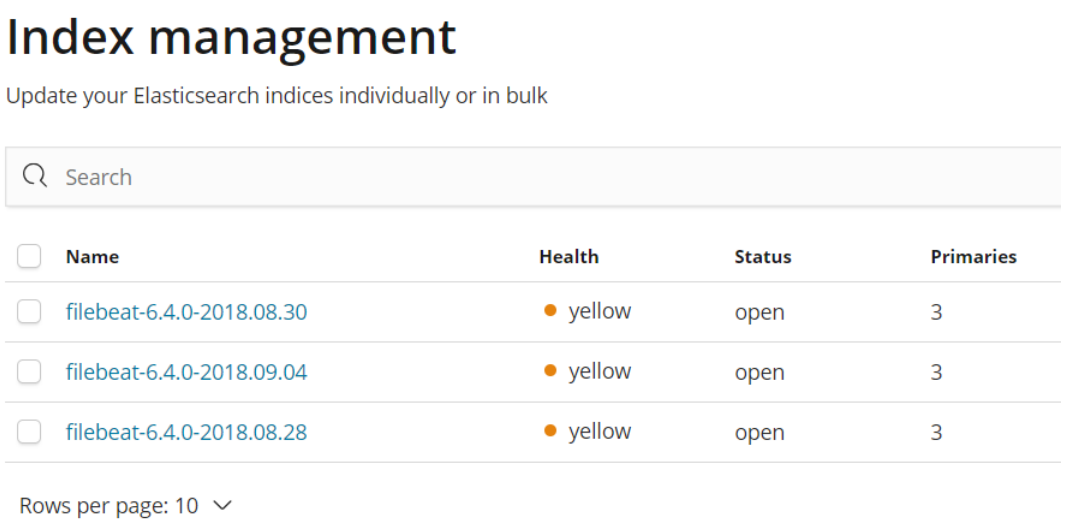
Create index patterns

Elasticsearch creates indexes on the data that it stores. Before you can view your PAS for OpenEdge logs in Kibana, you must define index patterns that match the index names of your log data in Elasticsearch.

To see a list of all Elasticsearch indexes, click **Management** in the Kibana web application's left pane and then **Index Management**.



Because your log data is sent by Filebeat, Elasticsearch creates index names with the pattern `filebeat-<filebeat_version>-<date>`.

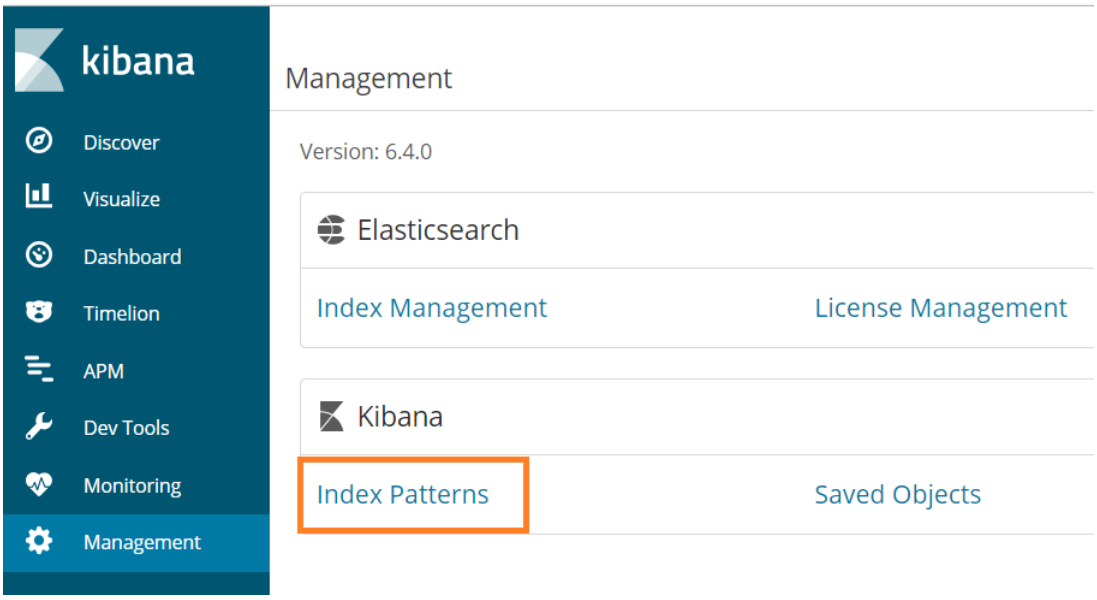


To view your PAS for OpenEdge logs, you must specify index patterns that match one or more of these indexes.

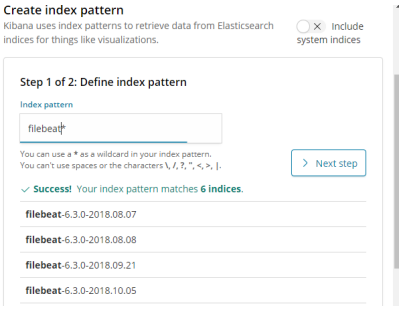
Depending on your needs, you may want to create multiple index patterns. For example, `filebeat-*` would fetch all Filebeat-based indexes, while `filebeat-6.4.0-2018.08-*` would fetch all Filebeat-based indexes created in the month of August, 2018.

Follow these steps to create an index pattern in Kibana:

1. Click **Management** in the left pane and then **Index Patterns**.



2. On the **Index Patterns** page, click **+Create Index Pattern**.
3. Enter the pattern in the **Index pattern** field. Kibana displays all indexes that match the pattern right below your entry.



4. Click **Next step**.
5. In the **Configure settings** page, select a **Time Filter** field.

Note: The **@timestamp** option enables you to filter log messages by time intervals. This is useful when you are troubleshooting.

---

6. Click **Create index pattern**.

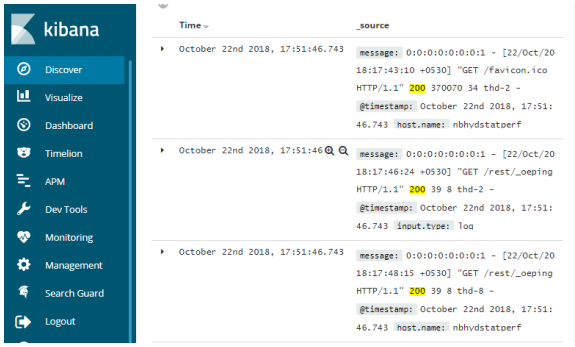
View PAS for OpenEdge logs

The **Discover**, **Visualize**, **Dashboard**, and **Timelion** features in the left pane enable you to view and analyze your log data in different ways.

Example

The Kibana documentation provides detailed descriptions of each feature but here is an example of how you could use the Discover feature to help you get started.

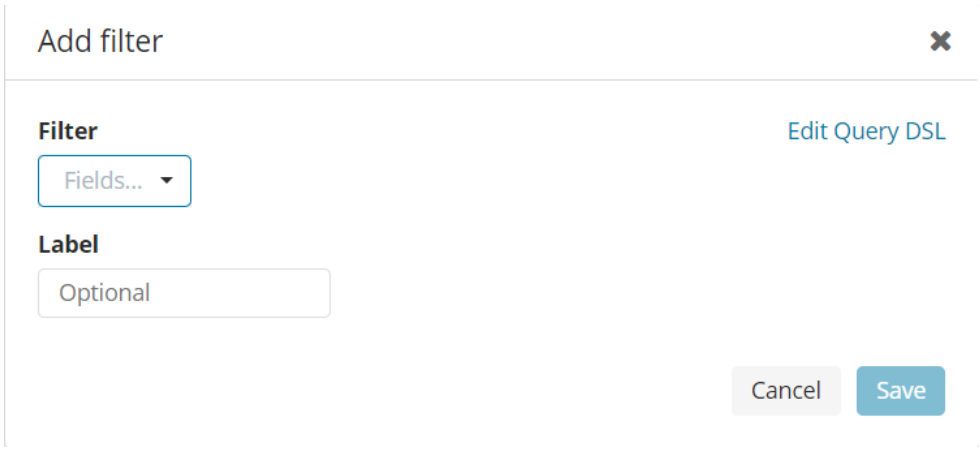
Click **Discover** and select an index pattern from the dropdown on the top left (under **Add a filter**). Based on the settings, all log messages that map to the index pattern are displayed.



A list of all available fields is displayed on the left. The **message** field contains a log message from one of your log files. Let's assume that you are troubleshooting and want to search for a particular type of message. For example, all messages that have the following stack trace line:

```
java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1067)
```

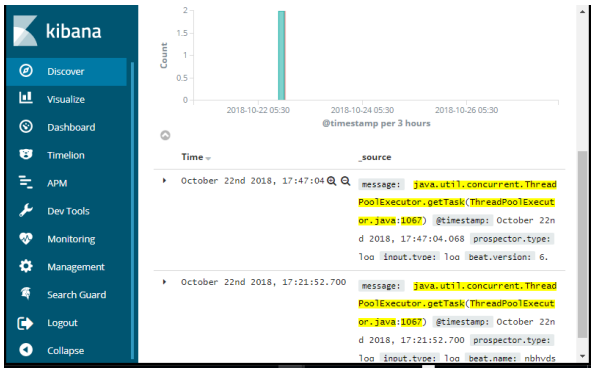
To do this, you need to specify a filter query. Elasticsearch provides a JSON-based query domain specific language (DSL) that enables you to write filter queries. Click **Add a filter+** and then **Edit Query DSL**.



Enter the JSON query, specify a label to identify it, and click **Save**:

```
{
  "query": {
    "match_phrase": {
      "message":
"java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1067)"
    }
  }
}
```

Only those log messages that fall within the selected index pattern and that pass the filter query are displayed:



You can drill down into each log message, add multiple filters based on a wide range of fields, and perform a host of other analytical and troubleshooting operations.

Additional resources

For more information on the visualization capabilities of Kibana, see the Kibana User Guide.

For more information on the Elasticsearch query language, see the Query DSL documentation.

Enhanced Mapped Diagnostic Content logging in PAS for OpenEdge

Progress has implemented the Logback framework for enhanced Mapped Diagnostic Content (MDC) logging. MDC allows the PAS for OpenEdge instances to dynamically inject request context information into the logging subsystems, where the request contexts can be exposed via tokens in log entry formats.

Tomcat logging states that each deployed application creates and maintains its own log file in a single file system directory. A PAS for OpenEdge is similar to one or more multi-threaded ABL application servers embedded into a single multi-threaded Tomcat instance. Therefore, PAS for OpenEdge client requests transition through multiple software components, and each produces related entries in multiple log files per component.

This design can hamper the troubleshooting process because of a lack of sub-component context. The lack of context stems from being unable to join the logging entries for a single request in a client session for a single server client through the request lifecycle. This makes PAS for OpenEdge log files difficult to use for anything other than failures that are self-describing in a single log entry.

With MDC logging, the PAS for OpenEdge default log entry formats can include essential request context information in all log files, as well as allow for user-made customizations when additional context is needed.

See the following topics for more information:

- Prerequisites for MDC Logging
- Configure MDC Logging
- MDC Logging Output Format
- Configure Tomcat Access Logs for MDC
- Configure ABL Spring and Session Manager Logs for MDC
- Configure ABL Agent Log for MDC
- Configure Spring Authentication and Authorization Audit Logs for MDC
- Configure MDC access to HTTP Request Headers

Prerequisites for MDC Logging

To use Enhanced MDC logging you must have the following components:

- PAS for OpenEdge release 11.7.4 or higher that provides the core MDC functional support for Security and ABL application.
- PAS for OpenEdge release 11.7.4 or higher oeabl web application war files that include MDC updates in `WEB-INF/logging.xml` and `WEB-INF/spring/**`.
- The file `conf/logging-security.xml` in your PAS for OpenEdge instance.
- An updated target PAS for OpenEdge instance `bin/openedge_setenv.{sh|bat}` file with the PAS for OpenEdge product update found in `DLC/servers/pasoe/bin/openedge_setenv.{sh|bat}`.

Configure MDC Logging

To begin using MDC Logging, select a set of enhancements you want to incorporate into your operations and configure the desired component or components. We have provided a few different business cases and the components that must be configured to enable MDC Logging:

Use case	Configure component
Add client request tracking	Tomcat Access Log ABL Spring and Session Manager Log ABL Agent Log

Use case	Configure component
Track Spring authenticated user-Id	Tomcat Access Log
Track user login and URL access control	Spring Authentication and Authorization Audit Logs
Track client HTTP request header fields	Tomcat Access Log
Add user login context from HTTP request	Spring Authentication and Authorization Audit Logs

MDC Logging Output Format

The standard output format of an MDC log across components consists of 3 colon-separated values in the following order: **webapp:transport:requestId**. The `requestId` value is left-padded hex, so an example output could be the following: **ROOT:w:0000000a**.

For the transport, the following abbreviations are used to determine the source:

- W = Web
- R = REST
- A = APSV
- F = File

For any request, the Tomcat Access log outputs this format exactly as stated if the pattern was modified. However, any message or standard output within the ABL Agent log is preceded by **R-** when the feature is enabled. If the message only comes from the internal Multi-Session Agent code then the output appears as **R-?:?:?** or simply **R-mtapsv:-:?**. When a message is output during execution of code from a transport request, then OERequestInfo is available and the expected values are provided for webapp, transport, and request ID.

Configure Tomcat Access Logs for MDC

Client request context information is injected into the Tomcat Access log entry format using existing Extended Access Log valve functionality that provides access to HTTP Request object attributes.

To enable the Tomcat Access logs for MDC values:

1. Open the `conf/catalina.properties` file for the target instance.
2. Find the `psc.as.accesslog.pattern` property which looks similar to the following codeblock:

```
psc.as.accesslog.pattern=%h %u %t "%r" %s %b %D %I %S
```

3. Edit the property to reference MDC fields for User-ID or Request-ID, as seen in the following example:

```
psc.as.accesslog.pattern=%h %{OEReq.userId}r %t "%r" %s %b %D %I %S %{OEReq.requestId}r
```

Note: The properties are case sensitive.

PAS for OpenEdge supports the following MDC fields:

MDC field	Description
`%{OEReq.requestId}r`	Track client HTTP requests through the Tomcat, ABL Session Manager, and Multi-session agent.
`%{OEReq.userId}r`	Replaces the Tomcat logging token %r with the OpenEdge Spring authentication user-id
`%{<http-header>}r`	Any standard Tomcat HTTP request headers can be included into your access log. See Configure MDC access to HTTP Request Headers for more information.

Configure ABL Spring and Session Manager Logs for MDC

The Logback logging framework supports using MDC tokens in the logging configuration layout pattern found in *web-app-name*`/WEB-INF/logging.xml`.

To enable the MDC values for ABL Spring and Session Manager, follow these instructions:

1. Locate the `webapps\`*`web-app-name`*`\WEB-INF\logging.xml` file for your target PAS for OpenEdge application.
2. Locate the logging pattern property which looks like the following codeblock:

```
<pattern>%d{HH:mm:ss.SSS}/%r [%thread] %-5level %logger{36} - %msg%n</pattern>
```

3. Edit the property to include desired MDC values, as seen in the following example:

```
<pattern>%d{HH:mm:ss.SSS}/%r [%thread] %-5level %X{OEReq.requestId} %logger{36} -
%msg%n</pattern>
```

   For more information on supported MDC field names, see the table below.

4. Restart your web application.
5. Edit the `logging.xml` file for any other target web applications.

Supported HTTP Request MDC Field Names

The following table shows all the MDC field names you can insert into the Spring and Session Manager logs:

Logging Field Name	Logging Field Description	Tomcat Access Log Syntax	ABL Application Log Syntax
`OEReq.userId`	The Spring Security authenticated user-id who is identified as the client from the server.	`%{OEReq.userId}r`	`%X{OEReq.userId}`
`OEReq.sessionId`	The HTTP Session ID assigned to the current PAS for OE client login session (if server login sessions are configured)	`%S`	`%X{OEReq.SessionId}`
`OEReq.requestId`	The PAS for OE assigned request ID relative to the server instance (Note: This property is not unique across load balanced server instances.)	`%{OEReq.requestId}r`	`%X{OEReq.requestId}`
`OEReq.webapp`	The Java web application context name the PAS for OE client request executes in	n/a	`%X{OEReq.webapp}`

Logging Field Name	Logging Field Description	Tomcat Access Log Syntax	ABL Application Log Syntax
`OEReq.transport`	The {apsv | soap | rest | web | file} PAS for OE service interface transport name the client's request executes in	n/a	`%X{OEReq.transport}`
`OEReq.remoteHost`	This field contains the remoteHost DNS address obtained from the PAS for OE network connection	`%h`	`%X{OEReq.remoteHost}`
`OEReq.remoteAddr`	This field contains the remoteHost IP address obtained from the PAS for OE network connection	`%a`	`%X{OEReq.remoteAddr}`
`OEReq.serverName`	This field contains the name of the PAS for OpenEdge server	`%v`	`%X{OEReq.serverName}`
`OEReq.URI`	This field contains the HTTP request's relative URI within `OEReq.webapp`	n/a	`%X{OEReq.URI}`
`OEReq.protocol`	This field contains the HTTP request scheme (http / https)	`%H`	`%X{OEReq.protocol}`
`OEReq.method`	This field contains the HTTP request's method name	`%m`	`%X{OEReq.method}`
`OEReqHead.`**`xxxxxx`**	This field contains the value of the HTTP request Header named **xxxxxx** or n/a if the header does not exist.	`%{xxxxxx}i`	`%X{OEReqHead.`**`xxxxxx`**`}`

Formatting Options for MDC values in ABL Spring and Session Manager logs

MDC values can be written to print a certain value if the requested information is not available. This is useful for troubleshooting. If an error occurs before the request ID, or other MDC field, is created, then the log can print a specified value.

The formatting options are written in this format:

```
%X{<name>[:-<default>]}
```

For example:

`%X{OEReq.requestId}`	The log shows the assigned request ID or blank if the logging occurs before a request ID can be created.
`%X{OEReq.requestId:--}`	The log shows the assigned request ID or "-" if the logging occurs before a request ID can be created.
`%X{OEReq.requestId:-na}`	The log shows the assigned request ID or "na" if the logging occurs before a request ID can be created

For more information, see the Logback layout and pattern documentation.

Configure ABL Agent Log for MDC

The only supported MDC field that is passed to the ABL Agent for logging is Request Id. This field is present in the PAS for OpenEdge multi-session agent via the currentRequestInfo attribute Request Id.

In OpenEdge 11.7.4 and higher, the request-id MDC field information is not automatically added to the PAS for OpenEdge Agent log file. It must be turned on after any adjustments are made in ABL log file parsers are updated.

To enable the request Id to be sent in ABL Agent logging, edit the `bin\openedge_setenv` script in the following manner:

- For Unix:

```
# Turn on extended client request ID logging in the
# multi-session Agent
REQIDLOGGING=y ; export REQIDLOGGING
```

- For Windows:

```
rem Turn on extended client request ID logging in the
rem multi-session Agent
set REQIDLOGGING=y
```

The ABL language has access to the MDC request-id field information via the `SESSION:CURRENT-REQUEST-INFO:REQUESTID` attribute. It does not require modifications to the `bin/openedge_setenv.{sh|bat}` file to use the MDC request-id field in your ABL application logging.

Configure Spring Authentication and Authorization Audit Logs for MDC

There are two Spring audit logs: Spring Authentication and Spring Authorization. The purpose for these logs is to independently capture authentication failures and successes, as well as URL authorization failures and successes. They are independent from the Spring runtime process logging that is found with the ABL Session Manager. They can be turned on without having to record Spring runtime logging. They are normally turned off and must be explicitly turned on.

Each deployed ABL application has its own set of authentication and authorization audit logs.

To turn on Spring Authentication and Authorization MDC logging, follow these instructions:

1. Verify that each target application has the following entry in the `webapps/<appname>/WEB-INF/logging.xml` file:

```
<include optional="true" file="${catalina.base}/conf/logging-security.xml" />
```

2. Locate the `conf/logging-security.xml` for your target instance.
3. Edit the `logging-security.xml` file to turn Spring Security client authentication and URL authorization events on.

   The default event logging levels are **OFF**, which means no logs files are created.

   The properties are displayed below with emphasis added on the value that needs to be edited:

```
<property scope="system" name="oeLogging.audit.authn.level" value="OFF" />
<property scope="system" name="oeLogging.audit.authz.level" value="OFF" />
```

   The valid values are written in the following table:

Value	Description
OFF	No security event logging will be recorded, nor will a log file be created. This is the default value.
ERROR	Record failure information only.
INFO	Record both failure and success information.

4. After the Spring Authentication and Authorization logs are set to **ERROR** or **INFO**, set the MDC field values as needed.

   Locate the **authn** (authentication) and **authz** (authorization) pattern elements in the `logging.xml`. These are two independent logging categories. Find the **authn** logging pattern, which looks similar to this:

```
<pattern>%d{HH:mm:ss.SSS}/%r [%thread] %-5level %logger{36} - %msg%n</pattern>
```

5. Edit the values to incorporate MDC field names, as in the following example:

```
<pattern> %d{HH:mm:ss.SSS}/%r ${contextName} %X{OEReq.requestId} %X{OEReq.remoteHost}
 %X{OESec.eventStatus} %X{OESec.loginPrincipal} - %X{OESec.failReason} -
%X{OESec.failDetails}%n
</pattern>
```

6. Edit the **authz** pattern in a similar manner.

   The support MDC fields for each logging pattern are described below.

Supported MDC field names for Spring Authentication and Authorization logs

The Spring Authentication and Spring Authorization audit logs monitor for different events.

Spring Authentication logging fields

The list of critical information fields related to Spring Security Authentication login pass/fail events includes:

MDC field name	Logging Field Description	ABL Application Log Syntax1
`OESec.eventType`	The type of Spring Security process being reported: "Authentication"	`%X{OESec.evnetType}`
`OESec.eventStatus`	The results of the client authentication process: { "success" | "failure" }	`%X{OESec.eventStatus}`
`OESec.loginPrincipal`	The client asserted account name for authentication	`%X{OESec.loginPrincipal}`
`OESec.failReason`	A simple string holding the reason for the authentication failure	`%X{OESec.failReason}`
`OESec.failDetails`	A more verbose string hold additional failure information	`%X{OESec.failDetails}`
`OEReq.xxxxxx`	The value of the HTTP Request.	`%X{OEReq.xxxxxx}`
`OEReqHead.xxxxxx`	The value of the HTTP Request Header.	`%X{OEReqHead.xxxxxx}`

Note: For more information about `OEReq` and `OEReqHead` MDC field names, see the **Supported HTTP Request MDC Field Names** table in the Configure ABL Spring and Session Manager Logs for MDC topic.

Spring Authorization logging fields

The list of critical information fields related to Spring Security URL authorization includes:

MDC Field Name	Logging Field Description	ABL Application Log Syntax1
`OESec.eventType`	The type of Spring Security process being reported: "Authorization"	`%X{OESec.eventType}`
`OESec.eventStatus`	The results of the client authentication process: { "granted" | "denied" }	`%X{OESec.eventStatus}`
`OESec.authzUserId`	The authenticated client's account name whose ROLEs are tested for URL access	`%X{OESec.authzUserId}`
`OESec.failObj`	The URL resource name being authorized	`%X{OESec.failObj}`
`OESec.failAttrs`	The authenticated client's account attributes used for authorizing URL access	`%X{OESec.failAttrs}`

MDC Field Name	Logging Field Description	ABL Application Log Syntax1
`OEReq.xxxxxx`	The value of the HTTP Request.	`%X{OEReq.xxxxxx}`
`OEReqHead.xxxxxx`	The value of the HTTP Request Header.	`%X{OEReqHead.xxxxxx}`

Note: For more information about `OEReq` and `OEReqHead` MDC field names, see the **Supported HTTP Request MDC Field Names** table in the Configure ABL Spring and Session Manager Logs for MDC topic.

Note: HTTP Request header fields are also supported for each audit log. For more information, see Configure MDC access to HTTP Request Headers.

Configure MDC access to HTTP Request Headers

Access to HTTP Header values as MDC fields can be set using the `OEReqHead.xxxxx` token, where `xxxxx` is the case-insensitive HTTP header name. The OpenEdge MDC support does not automatically make all HTTP headers available as MDC values. You must configure the list of header names if that level of information is required in your log files.

To configure the list of HTTP headers available as MDC fields:

1. Edit the PAS for OpenEdge instance `conf/oeablSecurity.properties` file, so the list is available to all deployed ABL applications and their ABL webapps.
2. Append the `MDC.filter.headerList` property with a comma separated list of header names:

```
MDC.filter.headerList=<http-header-list>
<http-header-list> := <header-name>[,<header-name>[,...]]
<header-name> := <case-insensitive header-name>
```

3. Save the `oeableSecurity.properties` file. The HTTP Request Header name is available for use in `OEReqHead` fields, as described in Configure Tomcat Access Logs for MDC, Configure ABL Spring and Sesson Manager Logs for MDC, and Configure Spring Authentication and Authorization Audit Logs for MDC.

Docker Container

PAS for OpenEdge is now available as a container so developers can develop, deploy, and run applications in an isolated container environment. For more information on the PAS for OpenEdge container, see the OpenEdge Container Project available on the Progress Community.