Progress®



OpenEdge® Service Pack 11.7.2:
New Information

OpenEdge®

Copyright

© 2017 Progress Software Corporation and/or one of 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, Making Software Work Together, 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, JustCode, JustDecompile, JustMock, JustTrace, 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 acknowledgments required to be provided in the documentation associated with the Progress product.

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 2017

Last updated with new content: Release 11.7.2

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....9
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....13
OpenEdge messages....13
Obtain more information about OpenEdge messages....14

What's New in 11.7.2....15

Installation and Configuration....15
Startup Parameters....16
Omit Log Messages (-omitLgMsgs)....16
Limit log file payload (-limitLgPayload)....17
Business Entity Support for Data Objects....18
Client formatting for properties of a JFP object....19
Handling SERIALIZE-NAME for fields referenced by a JFP....19
OpenEdge DataServers....20
OpenEdge Management....20
OpenEdge RDBMS....21
CDC User Identity....21
Database log file customization....22
OpenEdge Replication....23
Logging level for Replication....23
Transition End Trigger....24
OpenEdge SQL....26
Progress Application Server for OpenEdge....27
Configure applications to use the OpenEdge Authentication Gateway....27
ABL application PING service....32
Authenticate using OAuth2 and JWT....35
Extend OpenEdge SSO to web applications....51
Progress Developer Studio for OpenEdge....70

Server Technologies....70
Working with OpenEdge JMS Adapter 71

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 11.7.2: New Information documents important information about the OpenEdge Release 11.7.2 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 11.7.2 as the most recent Service Pack for Release 11.7.

Organization

This book contains the following major sections:

- Installation and Configuration on page 15

  Contains information about new and updated third party software included in this service pack.

- Startup Parameters on page 16

  Contains information about new startup parameters.

- OpenEdge ABL

  Contains information about updates to ABL features.

- OpenEdge DataServers on page 20

  Contains information about an update to the DataServer for MS SQL Server.

- OpenEdge Management on page 20

  Contains information about updates to OpenEdge Management.

- OpenEdge RDBMS on page 21

  Contains information about updates to the RDBMS.

- OpenEdge SQL on page 26

  Contains information about password security with SQL Explorer.

- Progress Application Server for OpenEdge on page 27

  Contains information about new features in PAS for OpenEdge.

- Server Technologies

  Contains information about a new bean property.

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**.

Convention	Description
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.
`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 11.7.2

This topic describes the new features included in OpenEdge Release 11.7.2.

For details, see the following topics:

- Installation and Configuration
- Startup Parameters
- Business Entity Support for Data Objects
- OpenEdge DataServers
- OpenEdge Management
- OpenEdge RDBMS
- OpenEdge Replication
- OpenEdge SQL
- Progress Application Server for OpenEdge
- Progress Developer Studio for OpenEdge
- Server Technologies

Installation and Configuration

Installing the OpenEdge Release 11.7.3 Service Pack includes the following updates:

- **Infragistics** — OpenEdge uses Infragistics NetAdvantage for .NET v2016 Vol 2. The Infragistics controls are upgraded to version 16.2.20162.2182.
- **Progress Compilation Tools (PCT)** — PCT is a set of ANT libraries, designed to automate work in the OpenEdge environment. PCT helps to automate build generation in a continuous integration process for OpenEdge applications.

  With PCT, you can automate certain tasks, such as:

  - Compiling and running ABL files
  - Database related tasks
  - Running unit tests
  - Generating API documentation
  - Generating deployable artifacts

Note: Progress ships PCT with the following OpenEdge products: OpenEdge Advanced Enterprise RDBMS, OpenEdge Enterprise RDBMS, OpenEdge Personal RDBMS, OpenEdge Workgroup RDBMS, 4GL Development System, OpenEdge Application Server Basic, OpenEdge Development Server, OpenEdge Application Server Enterprise, OpenEdge Auth Gateway, Progress Dev AS for OpenEdge, Progress ProdAS for OpenEdge, Visual Translator, Query/RESULTS, OpenEdge DataServer for Oracle, OpenEdge DataServer for MS SQL, WebSpeed Workshop, Progress Developer Studio for OpenEdge, OpenEdge Studio, Client Networking.

The PCT Version included in this service pack is 208.

When installed, PCT is located in *`install-dir`*`\pct\PCT.jar`

Refer to https://github.com/Riverside-Software/pct/wiki for detailed documentation of PCT.

Startup Parameters

The OpenEdge Release 11.7.2 Service Pack includes the following new startup parameters.

Omit Log Messages (-omitLgMsgs)

Use Omit Log Messages (`-omitLgMsgs`) to specify up to 16 messages you do not want written to the database log file.

Operating system and syntax	UNIX / Windows	`-omitLgMsgs` $n_1[,n_2[,n_x...[,n_{16}]]]$		
Use with	**Maximum value**	**Minimum value**	**Single-user default**	**Multi-user default**
Database Server	—	—	—	—

n

Message number(s). Specify up to 16 message numbers in a comma-separated list, with no embedded spaces in the list.

Fatal error messages cannot be suppressed with this parameter. The message numbers are not validated at startup, and invalid message numbers are ignored.

The message omission list is on a per database basis, and only applies to messages being written to the database `.lg` file. Messages written to the screen are not suppressed.

The values specified at startup are available via the _DbParams VST. The values can be read and updated.

To change this parameter while the database is running, use PROMON or the _DbParams VST.

You can use this parameter to support continuous operation of the database. For more information, see Change database parameters while the database is running.

Limit log file payload (-limitLgPayload)

Use Limit log file payload (`-limitLgPayload`) to prevent message payload from being written to the database log file. Message *payload* is defined as the user name, tty name, IP address, and connection hostname in a message.

Operating system and syntax	**UNIX / Windows**	**`-limitLgPayload` *n***		
Use with	**Maximum value**	**Minimum value**	**Single-user default**	**Multi-user default**
Database Server	1	0	0	0

n

Specify if payload is included or omitted.

- **0** — Limit payload is off; the complete contents of messages is written to the database log file.
- **1** — Limit payload is on; payload data within a message is not written to the database log file, and is replaced with the string, "<omitted>".

All messages written to the database log file (*dbname*`.lg`), and messages that are written to both the log file and the screen, honor the `-limitLgPayload` setting. Messages only written to the screen, do not check the payload setting. Other log files, such as for SQL or Replication, honor the payload setting when the information written to the log file is obtained from the database connection information.

The following examples displays the difference in messages written to the database log file when `-limitLgPayload` is enabled and disabled.

Table 1: Limit Payload Examples

Command or action	Limit payload enabled	Limit payload disabled
Server startup	`(4281) Server started by <omitted> on <omitted>.`	`Server started by jdoe on /dev/pts/17.`
Login	`(452) Login by <omitted> on <omitted>.`	`(452) Login by jdoe on /dev/pts/9.`
Logout	`(453) Logout by <omitted> on <omitted>.`	`(453) Logout by jdoe on /dev/pts/9.`

Remote login	`(742) Login usernum 24, userid <omitted>, on <omitted>.`
	`(7129) Usr 24 set name to <omitted>.`
	`(17961) User 24 set tty to <omitted>.`
	`(7129) Usr 24 set name to <omitted>.`
		`(742) Login usernum 24, userid jdoe client type ABL , on oesol64 108 using TCP/IP IPV4 address 172.16.18.108.`
		`(7129) Usr 24 set name to jdoe.`
		`(17961) User 24 set tty to oesol64 108.`
		`(7129) Usr 24 set name to jdoe.`

The value specified at startup is available via the _DbParams VST. The values can be read and updated.

The value specified can be viewed and modified in PROMON.

- To view the current setting, select from the **Promon Main Menu**, **R&D** > **4 (Administrative Functions)** > **7 (Server Options)**
- To change the current setting, select from the **Server Options** menu, **9 (Limit .lg message payload)**. The value is toggled between **Enabled** and **Disabled**.

To change this parameter while the database is running, use PROMON or the _DbParams VST.

You can use this parameter to support continuous operation of the database. For more information, see Change database parameters while the database is running.

Business Entity Support for Data Objects

OpenEdge Release 11.7.2 Service Pack includes the following updates on how the *`filter`* parameter to the Data Object Read operation is specified as a JSON Filter Pattern (JFP) object:

- Additional documentation on the format that a Kendo UI® client uses to specify property values in the JFP object
- Additional schema support used by the JFP object itself for temp-table fields defined with the `SERIALIZE-NAME` option

This information updates the topic, "Understanding the JSON Filter Pattern (JFP)," in the "Data Object Services" chapter of *OpenEdge Development: Web Services*, which describes the format and usage of the `filter` parameter to the Data Object Read operation when specified as a JFP object. This JFP format is a JSON object with a set of properties that specify how the data is to be filtered and returned by a Read operation that is invoked by a client using the JSDO dialect of the Kendo UI DataSource.

Client formatting for properties of a JFP object

The client DataSource initially sets values for two properties of the JFP object that it passes in a call to the JSDO `fill( )` method, which invokes a Data Object Read operation on the server. The DataSource initially sets these properties using a format specified in the Kendo UI DataSource documentation as follows:

- **`ablFilter` property** — Where `ablFilter` corresponds to the `filter` property that the DataSource passes in a call to `fill( )`. For more information on the format of this `filter` property value, see the DataSource `filter` configuration property description.
- **`orderBy` property** — Where `orderBy` corresponds to the `sort` property that the DataSource passes in a call to `fill( )`. For more information on the format of this `sort` property value, see the DataSource `sort` configuration property description.

For example, these properties can be passed in a call to `fill( )` like this:

```
clientJsdo.fill({filter: {field: "CustNum", operator: "gt", value: 5},
                 sort: { field: "Name", dir: "desc" }})
```

Where `clientJsdo` is a reference to a client JSDO with access to an OpenEdge Data Object resource whose ABL Read operation handles a JFP object parameter. Note that the `filter` and `sort` properties that the DataSource passes to `fill( )` are set from corresponding configuration properties of the same name that are previously set in the Kendo UI DataSource. The client call to the Read operation then converts the Kendo UI format of these property values to an equivalent ABL format as documented for the Business Entity method that implements the operation.

For more information on the JSDO `fill( )` method, see fill( ) method.

Handling SERIALIZE-NAME for fields referenced by a JFP

When you specify the `SERIALIZE-NAME` option for a temp-table field defined in a Business Entity, the schema for the field specified in the generated Data Service Catalog identifies the field by the alternative name specified using `SERIALIZE-NAME` instead of the original field name specified in the temp-table definition. For example, consider the following simple temp-table definition with one field:

```
DEFINE TEMP-TABLE ttCustomer BEFORE-TABLE bttCustomer
    FIELD Cust-Num AS INTEGER INITIAL 0  SERIALIZE-NAME "CustNum".
```

In this temp-table field definition, the field is defined in the generated schema with the alternative name `CustNum` instead of `Cust-Num`. You might use this option, for example, to remove the hyphens from field names for clients, such as JavaScript clients, where the hyphen (`-`) is always interpreted as a minus sign.

In the prior OpenEdge release, without additional annotations, when the client invoked the Read operation for a Kendo UI DataSource, any field defined with `SERIALIZE-NAME` that appeared in the `ablFilter` or `orderBy` property value of its JFP object parameter was identified in these property values using the alternative field name specified by `SERIALIZE-NAME` instead of its original field name in the temp-table definition.

As a result, using this `ablFilter` or `orderBy` property value directly as an ABL search or sort expression returned an unknown field error. In this example, no temp-table (or database) field exists with the name, `CustNum`. In the prior release, you could avoid this error by including a field-level annotation on the temp-table definition to identify the original field name to return in properties of the JFP object.

In this release, the generated schema for any field defined with `SERIALIZE-NAME` automatically includes a Catalog property that identifies its original field name without the need for any additional annotations on your part. When a JFP object is returned, its `ablFilter` and `orderBy` property values always identify the field by its original temp-table field name, not the alternative name specified by `SERIALIZE-NAME`.

OpenEdge DataServers

OpenEdge Release 11.7.3 Service Pack includes the following improvement to the OpenEdge DataServer for MS SQL Server:

Performance Improvement for two level single-shot join query

Starting in this release, two level single-shot join queries are evaluated as join-by-server on the server side. Single-shot refers to the use of either FIRST or LAST criteria as part of join query. This could result in significant performance improvements for the two-level single-shot join queries.

An example of single-shot ABL is shown below:

```
FOR EACH customer, FIRST [or LAST] order OF customer
    WHERE state = “MA”:
DISPLAY Name.
END
```

OpenEdge Management

OpenEdge Release 11.7.3 Service Pack includes the following improvements to OpenEdge Management:

- **Removal of Customer Experience Improvement Program page** — In earlier OpenEdge releases, the **Customer Experience Improvement Program** page enabled Progress Software to collect data related to the usage trends and patterns of the product, and thereby improve the quality of the product. To eliminate collection of such data, this page has now been removed.

  This page has been removed from the OpenEdge Management initial configuration pages, and the option to enable or disable this feature has been removed from the **OpenEdge Management General Configuration** (**Options** > **General**) page.

- **Support to stop a PAS application agent** — This release provides the capability to stop a PAS application agent gracefully or forcibly. You can choose how you want to stop a PAS agent by specifying the wait time using the following options in OpenEdge Management:

  - **Wait to finish** — The time, in seconds, the agent waits to complete the requests that are currently running before stopping gracefully.

    After waiting for the specified time, if **Wait after stop** is set to 0, the agent is stopped forcibly.

  - **Wait after stop** — The time, in seconds, the agent waits after completing **Wait to finish** and before stopping forcibly.

For more information about these options, see Progress Application Server for OpenEdge on page 27. For more information about managing PAS application agents using OpenEdge Management, see *OpenEdge Management: Progress Application Server for OpenEdge Configuration*.

- **Support to view new Database Startup Parameters** — OpenEdge Management provides support to view the following new database startup parameters:
  - **Omit log messages (`-omitLgMsgs`)**
  - **Limit log file payload (`-limitLgPayload`)**
  - **Log file truncate time (`-lgTruncateTime`)**
  - **Log file truncate frequency (`-lgTruncateFrequency`)**
  - **Log file truncate size (`-lgTruncateSize`)**
  - **Buffer Hash Latch Factor (`-hashLatchFactor`)**
  - **Log file archive enable (`-lgArchiveEnable`)**
  - **Log file archive directory (`-lgArchiveDir`)**

  For more information about the new startup parameters, see Startup Parameters on page 16. For general information about OpenEdge database startup parameters, see *OpenEdge Data Management: Database Administration*.

OpenEdge RDBMS

This section describes the enhancements to the RDBMS in the OpenEdge Release 11.7.3 Service Pack.

CDC User Identity

OpenEdge provides the option of identifying the user making a change within a Change Tracking Table record.

The user's identity is stored in the `_User-Name` field of the `_Cdc-Change-Tracking` table. There is no schema change.

The user's identity is set to one of the following options:

- **None** — Does not record any user identity.
- **Database** — Records the user's identity based on the database connection used by the user.
- **Application** — Records the user's identity based on the application session used by the user.

Configure the recording of the user's identity with OpenEdge Management or with a program authorized and written to manipulate the `_Db-Options` table. For details configuring the user's identity with OpenEdge Management, see *Manage Database Change Data Capture with OpenEdge Management* .

Configuration information

When configuring the CDC User Identity, the following restrictions apply:

- If at least one Security Administrator exists, then only a Security Administrator can change the user identity recorded in the Auditing and CDC Change Tracking tables.

- If no Security Administrator exists, only users with appropriate access to the `_Db-Options` table can change the user identity recorded in Auditing or CDC Change Tracking tables.
- The Security Administrator, or user with `_Db-Options` access, can request any of the following options for how the user identity is recorded in the Change Tracking Table:
  - None (this is the default behavior).
  - The user identity with no domain information (shortened user identity), for example `user1`.
  - The user identity with the domain information (fully qualified user identity), for example `user1@companydomain`.

  Note: This is provided implicitly and cannot be overridden if the database is enabled for multi-tenancy.

  Where optional, the reporting of a shortened or fully qualified user identity is database wide. However, when both Auditing and CDC are enabled, you cannot configure the user identity as separate options for each feature.

  For Auditing and CDC, the interactions are as follows:

  - Auditing does not need to be enabled to specify the qualification of the CDC user identity reported.
  - CDC does not need to be enabled to specify the qualification of the Auditing user identity reported.
  - However, either Auditing or CDC must already be enabled to specify this option at all.

Changing Identity

Although the changes to the CDC identity can be made online, the change only takes effect at the next occurrence of a SETUSERID() / SECURITY-POLICY:SET-CLIENT() / SET-DB-CLIENT() implicit or explicit action. This behavior is consistent with auditing.

If application session identity is requested, and the application has not set application session identity for the client connection, then the database connection identity is recorded.

Related Information

For more information user identity, see:

- The Application Security section of *OpenEdge Programming Interfaces*.
- The Audit Security section of *Learn about Security and Auditing*.
- The Configuring and Implementing Authentication in OpenEdge section of *Learn about Identity Management*.

Database log file customization

Startup parameters let you tailor messages written to the log file. You can limit payload information for security, or prevent certain messages from being written. The parameters that control message logging are:

- **Limit log file payload (`-limitLgPayload`)** — Prevent message payload from being written to the database log file. Message *payload* is defined as the user name, tty name, IP address, and connection hostname in a message. For details, see Limit log file payload (-limitLgPayload).
- **Omit log messages (`-omitLgMsgs`)** — Specify up to 16 non-fatal messages you do not want written to the database log file. For details, see Omit log messages (-omitLgMsgs).

OpenEdge Replication

The OpenEdge Release 11.7.2 Service Pack includes the following updates for OpenEdge Replication.

Logging level for Replication

OpenEdge Replication allows you to set the logging level for both the server and the agent. The logging level can be set at startup through the properties files, and can be changed while the system is running through a utility. The messages generated are written to either the `repl.agent.lg` or the `repl.server.lg` file, based on where in the properties file the logging level is set, or which database the utility is run against. If a logging level is not specified, logging is off.

Logging Levels

The table that follows describes the logging levels.

Logging Level	Description
0 (None)	Logs no entries. This is equivalent to turning logging off.
1 (Errors)	Logs OpenEdge Replication error messages. This is the minimum amount of logging.
2 (Basic)	Logs "basic" information such as the start and end of transactions and connections as well as summary information. Errors logged at Level 1 are also logged. This is the suggested logging level.
3 (Verbose)	Log all messages and details, including the information logged at Level 1 and 2.

Configuring logging level in properties file

You can set your logging level when you configure your replication and agent(s) in the `repl.properties` file. For both the `[agent]` and `[server]` sections of your properties file, add the following:

```
logging-level=n
```

Where *n* is an integer between 0 and 3.

Setting logging level while Replication is running

You can set or change the logging level while Replication is running for both the Replication server and agents. Use the following command to change the level:

```
dsrutil dbname -C logginglevel new-logging-level
```

dbname

Name of the database; specify the source database for the server or a target database for an agent.

new-logging-level

The new logging level to set. It must be an integer between 0 and 3.

The new logging level remains in effect until you change it again online, or until Replication restarts. If the Replication server or agent is restarted, the logging-level value set in the `repl.properties` file is used.

Transition End Trigger

OpenEdge Replication supports a post-transition trigger. The transition end trigger is a user-created script that is configured to run at the end of transition. Executing a transition end trigger gives the database administrator the opportunity to insert additional instructions at a controlled point in the transition, such as directing applications to re-connect to a new source database. The trigger is supported by fail-over transition, when a source and target switch roles, and recovery transition, when a source fails and a target transitions to be the source for the other target.

Configuring transition trigger

The transition end trigger is configurable. You can add the transition end trigger property to the properties file of the source, target(s), or both. Indicate that you want the transition end trigger to execute by adding the following line to the `[transition]` section of your properties file:

```
transition-end-trigger=1
```

If not specified, the `transition-end-trigger` value is 0, indicating that no trigger is to execute. If set to a value other than 0 or 1, or not properly inserted in the `[transition]` section of the properties file, an error is written to the *dbname*`.repl.agent.lg` file.

If the `transition-end-trigger` is enabled on more than one database involved in the transition process, then the trigger may be executed multiple times, once for each participating database where it is enabled. Depending on your configuration, the transition end trigger may involve one database (a source or a target), two databases (one source plus a target), or three databases (one source plus two targets).

The trigger file is not validated prior to transition.

Transition trigger

The transition trigger is a script with very specific requirements. The file must be:

- Named `transProc` on Unix or `transProc.bat` in Windows.
- Located in the target database directory, the directory containing the *dbname*`.db` file

- Set as "executable" by your operating system

Caution: An OpenEdge server calling an external script is a potential security risk. Restricted access to the script is advised.

The trigger file has five input parameters, in the following order:

1. Current database name (the database the trigger is called from after transition)
2. Transition status (success/failure)
3. Database name (new source database name)
4. Host name (host for new source database)
5. Port (port name or number for database broker)

Note: All five input parameters are not always available. If the transition has failed, only the first two parameters (current database and status) are valid. In instances such as an offline transition, the host name and port number parameters of the new source, may not be available.

Once configured to execute, the transition end trigger executes regardless of the success or failure of the replication transition, by spawning a new process at the end of transition. Within the trigger code, you can use the transition status to determine execution paths based on the success or failure of transition.

The trigger file is not validated prior to transition. If the `transProc` file does not exist in the expected directory, or cannot be executed, an error message is written to the *dbname*`.lg` file.

If the trigger script is not found, the following error message is written:

```
(293) ** db_directory/transProc was not found.
```

If the trigger script is not executable, or you do not have permission, the following error message is written:

```
(294) ** You do not have permission to access file db_directory/transProc.
```

If the trigger script successfully executes, the following messages are written:

```
RPLU    6: (-----) Transition operation: Executing transition end trigger.
RPLU    6: (-----) Executing transProc with args[0]: /db_directory/transProc
RPLU    6: (-----) Executing transProc with args[1]: db1
RPLU    6: (-----) Executing transProc with args[2]: 0
RPLU    6: (-----) Executing transProc with args[3]: /db_directory/db2
RPLU    6: (-----) Executing transProc with args[4]: localhost
RPLU    6: (-----) Executing transProc with args[5]: 2625
RPLU    6: (13551) The transition of this database has completed normally.
```

Replication Monitor

You can verify the setting of `transition-end-trigger` property in the Replication Monitor.

For the server, select S for the **Replication server status** screen. The lower portion of the screen, with an enabled trigger, is shown below:

```
    AI structure file:                    addai.st
Backup method:                            Mark
Transition end trigger:                   1

Configured agents:                        agent1, agent2
Repl keep alive:                          300
Schema lock action:                       Wait
Agent shutdown action:                    Recovery
```

For an agent, select A for the **Replication agent status** screen. The lower portion of the screen, with an enabled trigger, is shown below:

```
Additional transition information:
    Replication set:                      1
    Database role:                        Reverse
    Transition to agents:                 agent1,agent2
    Restart after transition:             1
    Automatically begin AI:               1
    Automatically add AI areas:           1
    AI structure file:                    addai.st
    Backup method:                        Full-offline
    Transition end trigger:               1
```

OpenEdge SQL

The OpenEdge Release 11.7.3 Service Pack includes the following update for OpenEdge SQL:

Securely specifying a password in SQL Explorer

The purpose of SQL Explorer password security is to find a safe and secure way to provide password security.

While using SQL Explorer, the password is specified using the `-password` option which is in clear text and is visible. SQL Explorer password security helps in blocking this security breach. Using this feature, the password value cannot be viewed by UNIX operating system tools like `ps` and any other Windows equivalents.

For SQL Explorer password security, the options are (in the sequence of most secured to least):

- **Prompt for password** — Previously, the `-password` option of SQL Explorer was mandatory when it was used along with `-user` option. Now this is optional. The SQL Explorer utility prompts for password when it is not provided in the command line. While typing, the password is neither visible in the console and nor viewed by any operating system utility.
- **Using sqlexp in batch mode** — SQL user can redirect the output of `echo` / `cat` commands to `sqlexp` via 'pipe' ('|'). This option is less secure than the one above.
- **Using genpassword utility** — This option is for those who want to use the command line to provide the password. Use the `genpassword` utility to generate an encrypted password instead of using the actual password. Then use the encrypted password in the SQL Explorer tool.

If the password is `test` for user `userA` then use the `genpassword` utility to get an `OECH1` encrypted password for `test`, as shown:

```
genpassword -password test
encrypted-password
```

You can take the *`encrypted-password`* output from the `genpassword` utility and prefix with `oech1::`, and supply that to `-password` option of `sqlexp`.

For more information on the `genpassword` utility, see *OpenEdge Getting Started: Installation and Configuration*.

Progress Application Server for OpenEdge

This section describes the new features in PAS for OpenEdge that are included in the OpenEdge Release 11.7.3 Service Pack.

Configure applications to use the OpenEdge Authentication Gateway

Beginning in OpenEdge Release 11.7.2, the functionality of the OpenEdge Authentication Gateway was extended to be a Security Token Service (STS) for OpenEdge applications hosted on a PAS for OpenEdge instance. In prior releases, its use was limited to instances and database connections.

For more information about the basic architecture of the STS AuthenticationProvider and its configuration, see the following topics:

- The STS AuthenticationProvider on page 27
- Configuration and Testing on page 28

The STS AuthenticationProvider

The STS AuthenticationProvider in PAS for OpenEdge is a plug-in that enables the use of the OpenEdge Authentication Gateway as the source for user authentication and client-principal token generation.

The STS AuthenticationProvider:

- Takes user credentials (i.e. user name and password) as input
- Connects to the Authentication Gateway via a secure HTTPS connection
- Sends user credentials to the Authentication Gateway for authentication

If the authentication is successful, the STS AuthenticationProvider:

- Receives a sealed client-principal token from the Authentication Gateway
- Passes the client-principal (unaltered) to the Spring Security URL authorization process

Spring Security URL authorization:

- Performs Role-Based Authorization (RBA)

- Passes the client-principal token (if RBA is successful) to ABL applications

---

Note: The client-principal's `ROLES` attribute serves as input to the Spring Security URL RBA process. Therefore, the `ROLES` attribute must contain at least one role name that meets Spring Security's URL access control requirements.

Like other Spring Security components in PAS for OpenEdge, the STS AuthenticationProvider is configured in an `oeablSecurty.properties` file on the instance, ABL application, or web application levels.

For more information about the hierarchy of these `oeablSecurty.properties` files, see 'Enable ABL application authentication' in *Manage Progress Application Server (PAS) for OpenEdge*.

On the instance level (`../conf/oeablSecurity.properties`, the file contains a complete set of STS AuthenticationProvider properties. For example:

```
## OpenEdge Authentication Gateway client configuration
 ## for direct user logins to a PASOE server
 ##
 sts.AuthProvider.multiTenant=true
 sts.AuthProvider.userDomain=
 sts.UserDetails.stsURL=https://host:port
 sts.UserDetails.stsKeystore=
 sts.UserDetails.clientHeaderName=x-oests-token
 sts.UserDetails.noHostVerify=false
 sts.UserDetails.certLocation=${psc.as.oe.dlc}/certs
 sts.UserDetails.tlsCipherSuites=
 sts.UserDetails.tlsProtocols=
 sts.UserDetails.userAgent=PASOE (Spring)
 sts.UserDetails.sniHost=
```

The files on the ABL application (`../ablapps/`*`abl-app-name`*`/oeablSecurity.properties`) and the web application levels (`../webapps/`*`web-app-name`*`/WEB-INF/oeablSecurity.properties`) contain the properties that can override the properties specified on the instance level. For example:

```
## OpenEdge Authentication Gateway client configuration
 ## for direct user logins to a PASOE server
 ##
 sts.AuthProvider.multiTenant=true
 sts.AuthProvider.userDomain=
 sts.UserDetails.stsURL=https://host:port
 sts.UserDetails.stsKeystore=
 sts.UserDetails.noHostVerify=true
 sts.UserDetails.sniHost=
```

See the `../conf/oeablSecurity.properties.README` file for a description of the STS AuthenticationProvider properties and an explanation of their valid values.

Configuration and Testing

To use the OpenEdge Authentication Gateway as an STS authentication provider for a PAS for OpenEdge instance, you must configure the instance as an Authentication Gateway client. The following sections describe how to setup, configure, and troubleshoot an Authentication Gateway client:

- OpenEdge Authentication Gateway configuration on page 29
- Configure a web application on page 30
- Troubleshoot the PAS for OpenEdge instance on page 31

See OpenEdge Authentication Gateway utilities for information about the command line tools that are available to configure and manage the Authentication Gateway.

OpenEdge Authentication Gateway configuration

The OpenEdge Authentication Gateway must have a valid HTTPS TLS (Transport Layer Security, the successor to SSL) certificate and client-key installed.

If it hasn't been done already, an administrator can create a new STS Client Key file from an STS Server Key file using the OpenEdge STS Key Utility. For example:

```
oe_install_dir/bin/stskeyutil install -url https://oeag-dns-name:oeag-port-number
 -file oeagserverkey.ecp
```

Important: Always remove the `.ecp` file from the server after an install to prevent unauthorized access to your configuration. Store it in a secure location in case you need it to re-configure at a later date.

Note: Beginning in OpenEdge 12.3, STS client key management can be handled by the Key Distribution service. For more information, see About STS client key management.

Test the installation with the STS Client Utility. For example:

```
oe_install_dir\bin\stsclientutil -url https://yourmachinename:port -cmd ping
[-nohostverify]
```

Note:

Be aware of these differences between a development server (used for application development) and a production server (used for application deployment):

- For development servers, the Authentication Gateway and its clients ship with a server certificate specifically crafted to allow TLS connections, but which will fail post-connection Hostname validation. Therefore, the client connections to an Authentication Gateway always must include the `-nohostverify` option to work. For example:

  The `-nohostverify` option allows full server certificate validation per Public Key Cryptography Standards (PKCS) standards, but skips post-connection Hostname validation.

- For production servers, the Authentication Gateway server must obtain, configure, and use a non self-signed server certificate. Therefore, it is required that the ROOT CA and any optional Intermediate CA certificates needed to validate the Authentication Gateway server's certificate be installed into the OpenEdge installation's `DLC/certs` directory. Install the ROOT CA and Intermediate CA certificates using only the OpenEdge supplied `DLC/bin/certutil` utility. Any other installation mechanism is not guaranteed to work.

After verifying that the Authentication Gateway is running, test HTTP connectivity:

```
oe_install_dir/bin/stsclientutil -cmd ping -url
https://oeagd-dns-name:oeag-port-number [ -nohostverify ]
```

For more information about client-key generation, installation, and testing, see STS Client Key (for database server).

Configure a web application

To configure a web application to use the OpenEdge Authentication Gateway, set the following properties in the PAS for OpenEdge web application's *instance_name*/webapps/*Web_app_name*/WEB-INF/oeablSecurity.properties file:

1. Enable the use of an STS AuthenticationProvider by setting the `http.all.authmanager` to `sts`:

```
http.all.authmanager=sts
```

2. Specify the URL for the Authentication Gateway connection by setting the `sts.UserDetails.stsURL` property:

```
sts.UserDetails.stsURL=https://oeag-dns-name:oeag-port-number
```

3. (Optional) Enable the `-nohostverify` option to turn off host verification.

```
sts.UserDetails.noHostVerify=true
```

Note: `-nohostverify` is only for servers used for application development and testing.

4. (Optional) Specify a fixed OpenEdge domain name that is appended to the client's user ID before authenticating with the Authentication Gateway's STS.

```
sts.AuthProvider.userDomain=sts-configured-domain-name
```

5. (Optional) Specify the directory where the STS AuthenticationProvider looks for the Authentication Gateway's client key file:

```
sts.UserDetails.stsKeystore=client-key_pathname
```

The STS AuthenticationProvider searches `oeablSecurity.properties` files for a specified `sts.UserDetails.stsKeystore` value in the following order and uses the first value it finds:

a. $CATALINA_BASE/webapps/*web-app-name*/WEB-INF/
b. $CATALINA_BASE/ablapps/*abl-app-name*/conf/
c. $CATALINA_BASE/conf/
d. $CATALINA_HOME/conf/

If the path to the client key file is not specified in any of those `oeablSecurity.properties` files, the value of the STSKEYSTORE multi-session Agent process environment variable is used. And if STSKEYSTORE is not set, the default client key file in *openedge_install_dir*`/keys` is used.

6. (Optional) Use the sniHost property 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 your web application to connect to.

```
sts.UserDetails.sniHost=virtual-hostname
```

The web application requests the virtual host's TLS certificate during the TLS handshake instead of the Server URL's host.

Next, edit the web application's URL access control file, *instance_name*`/webapps/`*Web_app_name*`/WEB-INF/oeablSecurity.csv`. If your URL access controls are role-based (for example, when a user must be assigned a particular role to gain access), you must change the `hasRole(...)` field to include one of the role names inserted into a client principal issued by the Authentication Gateway's STS.

Note: If the web application's URL access controls use `hasRole(...)` and the client principal issued by the Authentication Gateway's STS does not include a role attribute, access to some of the application's URLs will be rejected.

Finally, optionally specify any advanced STS AuthenticationProvider properties, which include, but are not limited to:

- Customized TLC connection attributes
- A customized HTTP header name that is used to pass client-key authorization to an STS
- A customized HTTP User-agent header value that identifies the client to an STS

Detailed description of these advanced properties can be found in the *instance_name*`/conf/oeablSecurity.properties.README` file.

Troubleshoot the PAS for OpenEdge instance

After changing any Spring Security configuration in a PAS for OpenEdge instance, you should restart the instance and ensure that the changes are of the right data type, value, and definition. When the server completes its startup operations, check for errors in all of the log files, and determine if they are related to the Spring Security configuration changes. The Spring Security configuration errors are normally found in the instances's *instance_name*`/logs/localhost.`*date* log file. The error will always contain an extensive message indicating that Spring could not create a bean, and will be followed by a Java exception stack dump. The last part of the message will indicate which Spring bean failed to start and what the reason is.

Once the PAS for OpenEdge instance starts without errors, test user authentication and URL access by logging in to the web application.

If user authentication fails, investigate other Authentication Gateway connection failures or user authentication failures returned by the Authentication Gateway's STS. This level of information will typically be found in the instance's log files. If an access error (404/403) is returned to the PAS for OpenEdge client then look into the instance's log files for errors when checking the URL access controls against the STS issued Client-Principal Role attribute.

Advanced troubleshooting can be performed by enabling DEBUG logging in the web application's `WEB-INF/logging.xml` file. DEBUG settings that are specific to STS authentication are:

```
<logger name="com.progress.appserv.services.security.OESTSAuthProvider" level="DEBUG"/>
<logger name="com.progress.appserv.services.security.OESTSUserDetailsImpl"
level="DEBUG"/>
```

Other Spring Security authentication and URL authorization process errors can be discovered by enabling DEBUG logging:

```
<loggername="org.springframework[optional-package-path]" level="DEBUG" />
```

ABL application PING service

The PING service allows an ABL client to find out the state of a data service (online or offline) and the ABL application's ability to respond. This service can be used in conjunction with other static service resources (such as `/static/home.html`), and the HTTP status to determine whether the server, service, transport, and ABL application are available for use.

The APSV, REST, and WEB transports use the `ServerStatus()` method of the `OpenEdge.Rest.Admin.AppServerStatus` class to implement the PING service. You can extend the class to return application-specific information about database connections, PROPATH, loaded libraries, caches, available services, and more.

Note: The PING service is not available for the SOAP and Web transport.

Transport Syntax

REST

```
GET { http | https }://host:port/web_app/rest/_oepingService/_oeping
```

WEB

```
GET { http | https }://host:port/web_app/web/_oepingService/_oeping
```

To enable the PING service for the WEB transport, you must configure a handler in the `openedge.properties` file:

```
handlern=class name : /uri
```

Note: When a GET method is invoked on `/_oepingService/_oeping`, the `HandleGet` method in the class that is configured in the web handler is invoked. The response body in `HandleGet` is displayed on the web page as output.

The PING service only responds to the URL as it is configured in the `openedge.properties` file.

APSV

```
Run OpenEdge/ApplicationServer/Util/apsv_oeping.p on happsrv(OUTPUT
    JSON_Data).
```

happsrv is the application server handle, and *JSON_Data* is the JSON output.

JSON output format

When the PING service is successful, it can return a JSON array with the following information:

```
{“response”: {“_retVal”:  “text” }}
```

Note:

If nothing is returned to the PING service, then that indicates that the agent is available, since the default value for *text* is blank (" ").

To customize the return value, extract the `OpenEdge.Rest.Admin.AppServerStatus` class from the *oe-install-dir*`/src/OpenEdge.ServerAdmin.pl` procedure library, and place the customized version in your PROPATH.

If the PING service is not successful, then it returns a JSON array with the following information:

```
{"_errors": [ {"_errorMsg": "error_text",
        "_errorNum": error_number} ]}
```

Note that the error message may include additional properties.

Secure the PING service

You can restrict access to the PING service in the */web-app/*`WEB-INF/oeablSecurity.csv` file. For example, to restrict everyone from accessing REST PING, add the line in bold:

```
############## Intercept-url definitions for the REST transport URIs###############
"/rest/_oepingService/_oeping",”*”,"denyAll()"
"/rest/**","*","hasAnyRole('ROLE_PSCAdmin')"
    ###################################
```

Note: Order is important. The line with `denyAll()`, which restricts access to the PING service, must appear before the line that follows, which allows access to all REST services.

Example

The following code returns the server status that includes the names of connected databases and PROPATH entries.

```
method public character ServerStatus( ):
        define variable statusJson as JsonObject no-undo.
        define variable dataArray as JsonArray no-undo.
        define variable svrStatus as character no-undo.
        define variable cnt as integer no-undo.
        define variable maxCnt as integer no-undo.

        assign statusJson = new JsonObject().
          dataArray  = new JsonArray().
        statusJson:Add('ldb',dataArray).

        do cnt = 1 to num-dbs:
          dataArray:Add(ldbname(cnt)).
        end.

        assign maxCnt  = num-entries(propath)
          dataArray  = new JsonArray().
        statusJson:Add('propath', dataArray).
        do cnt = 1 to maxCnt:
          dataArray:Add(entry(cnt, propath)).
        end.

        statusJson:Write(output svrStatus).

        return svrStatus.
        end.
```

The output from the example code looks similar to the following:

```
{
        "response": {
              "_retVal": {
                    "ldb": [
                           "sports2020"
                           ],
                           "propath": [
        "c:\\username\\pas\\117\\baseinstall\\webapps\\ROOT\\WEB-INF\\openedge",

        "c:\\username\\pas\\117\\baseinstall\\openedge",

        "c:\\username\\dlc\\oe117\\tty",
        "c:\\username\\dlc\\oe117\\tty\\ablunit.pl",
        "c:\\username\\dlc\\oe117\\tty\\adecomm.pl",
        "c:\\username\\dlc\\oe117\\tty\\adecomp.pl",
        "c:\\username\\dlc\\oe117\\tty\\adeedit.pl",
        "c:\\username\\dlc\\oe117\\tty\\adeshar.pl",
        "c:\\username\\dlc\\oe117\\tty\\dataadmin.pl",
        "c:\\username\\dlc\\oe117\\tty\\OpenEdge.BusinessLogic.pl",

        "c:\\username\\dlc\\oe117\\tty\\OpenEdge.Core.pl",
        "c:\\username\\dlc\\oe117\\tty\\OpenEdge.ServerAdmin.pl",

        "c:\\username\\dlc\\oe117\\tty\\prodict.pl",
        "c:\\username\\dlc\\oe117",
        "c:\\username\\dlc\\oe117\\bin"
        ]
      }
    }
}
```

Authenticate using OAuth2 and JWT

Open Authorization (OAuth) is a standard framework that allows login access to third-party websites and applications without exposing user account credentials and information. Currently, OAuth2 is the latest version of that standard.

When a product is made up of components from multiple sources that must share a common user identity, OAuth2 and JSON Web Tokens (JWT) provide a way to support single user authentication to the product as a whole. Authentication results in a single sign-on (SSO) security token that allows access to all the product services that are enabled to use the token for authorization.

OAuth2, implemented correctly, provides a medium level of security for a broad range of web application architectures, including web browsers, mobile devices, browser-based client applications (such as JavaScript), and B2B client applications.

OAuth2 and JWT standards are enabled in PAS for OpenEdge to allow it to participate in SSO outside a homogeneous OpenEdge environment. This topic provides a general description of the OAuth2 and JWT standards, a description of how the standards are supported in PAS for OpenEdge's Spring Security framework, and, instructions about configuring and troubleshooting.

Topics include:

- About OAuth2 on page 35
- About OAuth2 security considerations on page 37
- About OAuth2 tokens on page 37
- About JSON Web Tokens on page 39
- About support for OAuth2 and JWT in PAS for OpenEdge on page 42
- Configure a PAS for OpenEdge web application as an OAuth2 Resource Server on page 44
- Debug logging for OAuth2 on page 51

About OAuth2

This topic introduces core OAuth2 concepts and terms. Watch the following video to learn about OAuth2 support in PAS for OpenEdge.

Use OAuth2 with the Progress Application Server for OpenEdge.

Actors

OAuth2 defines the following *actors* to perform specific operations and to interact to create a secure authorization framework:

Actor	Description	OpenEdge Examples
Resource owner	A user who is the owner of data stored, accessed through, and protected by a *resource server*. As a data owner, you can authorize a *client* to access and perform certain operations on data stored on a resource server.	End users

Actor	Description	OpenEdge Examples
Client	An application that uses an access token from an *authorization server* to gain access to the resource owner data that is hosted by a resource server. Each client is defined by a unique ID (`client_id`) that is registered with an authorization server, inserted into an OAuth2 access token, and validated by the resource server.	JavaScript client applications
Resource server	An abstraction that refers generally to any system or component that stores resource data and returns it to a client that is authorized to receive it. For example, the web server data service that hosts and authorizes client access to a resource owner's data based on an access token's client ID, user identity, and granted scope.	ABL business applications (Spring OAuth2 in OEABL web applications)
Authorization server	A server that validates a resource owner's credentials. Maintaining a list of registered clients and resource servers, it calls on an authentication server for identity verification. If authorization is successful, then it issues access tokens to a client on behalf of the resource owner.	N/A (third-party products)
User agent	A web browser, or some other type of framework that hosts client applications.	Browsers, mobile devices, Node.js, and so on
Authentication server	A server, undefined in the standard, that is used by an authorization server to authenticate the resource owner's identity. After the resource owner's identity is authenticated, the authorization server can continue its authorization process for issuing an access token.	N/A (third-party products associated with the authorization server)

Flows and grant types

A *flow* represents an exchange of one or more HTTP messages between the various OAuth2 actors. Each OAuth2 flow has a name defined in the standard as a *grant type*. The grant type is configured and implemented in a client and a authorization server, and may be included in an access token for validation by a resource server.

The grant types currently specified include:

- Authorization (code) grant

  A flow used by confidential clients. The client interacts with the resource owner and interfaces with an authentication server through a user agent. The resource owner authenticates to an authorization server that issues an authorization code to the client. The client then uses the authorization server to exchange its authorization code for an access token it can use to access the resource server on behalf of the resource owner

- Resource owner password (credentials) grant

A flow used in cases where the resource owner has a trust relationship with the client. An example of this type of relationship is a user (resource owner) with a mobile device (client). In this flow, the client and resource owner work together to authenticate to an authorization server that issues an access token to the client. The client manages the access token and uses it to access the resource owner's data on the resource server.

- Client credentials grant

  A flow that is used only when the client is a trusted application. For example, the client may be a partner company's business application that is also a resource owner. This is a very simple flow where the client authenticates to the authorization server using its own user ID and password and receives an access token in response. The client uses the access token to access data on a resource server.

- Implicit grant

  A flow often used by a public client. In this flow, the client initiates the flow by directing a user agent to an authorization server that interacts with and authenticates the resource owner. If the resource owner grants the client access to their data, the user agent is sent an access token, it can use on behalf of the client when the client accesses the resource server.

About OAuth2 security considerations

Some security experts think OAuth2 is not a secure mechanism for use in browsers, JavaScript applications, web servers, or web applications. However, you can use OAuth2 to reduce security vulnerabilities if you follow best practices. The basic best practices are:

- Securing authorization and resource server implementations using code reviewed, and scanned, implementations
- Ensuring that every HTTP message travels through TLS network connections
- Ensuring that cryptography keys are securely stored and shared between authorization and resource servers
- Fully validating an access token (according to its specifications) before it is used to access data

About OAuth2 tokens

In OAuth2, security tokens transport authorization information between clients, user agents, authentication servers, and resource servers. Tokens are opaque values in the OAuth2 standard and can be implementer defined. The two actors that must agree on a token's form and content are the authorization server that produces a the token, and the resource server that uses the token's value to grant or deny a client's access to resource owner data.

Note: The server that produces the OAuth2 token is a third-party implementation. The tokens cannot be generated from the OpenEdge environment

---

A token must be recognized by a server as one of these forms before it can be used:

- Authorization token
- Self-contained access token (also known as an ID token, or identity token)
- Random access token
- Refresh token
- Bearer token

Authorization tokenss

An *authorization token* is a random data value issued by an authentication server to a client that then sends it to an authorization server and exchanges it for an access token (self-contained or random).

Self-contained access tokens

An OAuth2 *self-contained access token* is a Base64-encoded data value that holds the resource owner's identity and granted access rights (also known as scope). This form of access token can be used directly by a resource server's authorization process to grant or reject access to services or data, because it contains the client identity, resource owner identity, and resource owner scope necessary to perform authorization. In most implementations the token is a JSON web token, but the OAuth2 standard does not specify the token's actual format. For more information, see About JSON Web Tokens on page 39.

Self-contained access tokens are commonly issued by enterprise authentication services to clients that can use the token in multiple resource servers. As explained in About OAuth2 on page 35, a resource server is an abstraction that refers generally to any system or component that stores resource data and returns it to a client that is authorized to receive it. Authorization is determined by the access token that is sent by the client. It is the responsibility of the resource server to limit the authentication services from which it accepts access tokens by validating the token before it may be used.

The general guideline states that an authorization server can issue a self-contained access token directly to trusted clients that send it to the resource server, in each service request, as a bearer token. The resource server's authorization process validates the self-contained access token's authenticity, and then uses the token's identity, or access right values, to grant or reject access and add to the service's audit trail.

Some authorization servers support a request from a resource server to exchange an untrusted client-supplied random access token for a self-contained access token that contains the resource owner's identity and granted scope. In such a case, the resource server logs in to the issuing authorization server and exchanges the random access token for a self-contained access token. This is a much more complex, non-standards-based process and is not widely available.

The method that is used by a resource server to obtain a self-contained access token is determined by the authorization server's configuration and implementation according to the type of registered client. In both use cases, the resource server must obtain and fully validate a self-contained access token for expiration and data integrity before it grants access to resource owner data.

Random access tokens

An OAuth2 *random access token* is a unique Base64-encoded random data value that is typically issued to untrusted clients. Random access tokens are commonly used when an authorization server coexists with one or more resource server services in the same product. The randomness of the access token's value reduces the risk of the resource owner's identity and access rights being stolen by the client application code. Random access tokens work because the authorization server *and* the resource server share the same (internet) network and storage of self-contained access tokens.

However, because the random access token does not contain a client identity, or a resource owner's identity, or granted data rights, a resource server cannot use the token to grant access to a resource owner's data. Before an exchange of random access tokens can take place, the resource server and authorization server must first establish a trust relationship whereby each trusts the other to not compromise any access tokens. The trust relationship can be established in several ways, and is not specified as part of the OAuth2 specification. Examples of a trusted relationship between an authorization server and resource server are the use of a SQL database, in-memory storage, or a fully authenticated HTTP login.

Refresh token

An OAuth2 *refresh token* is a unique Base64-encoded random data value that can (optionally) be issued by an authorization server in conjunction with a random or self-contained access token. To minimize the time a random or self-contained access token can be used after being stolen by a third party, an access token's expiration times can be kept very short. To not overly burden the resource owner with continually re-authenticating to obtain a new non-expired access token, an authorization server can be configured to issue a refresh token, which the client application can use to automate the process of obtaining a new access token from the authorization server when the current one has expired.

The automated OAuth2 refresh token process typically takes place in the client when the resource server returns an expired token status. The resource server detects access token expiration only when an expiration value is present in the token and returns only a token expired error to the client. The resource server does not play a role in the client obtaining a new access token. When the client detects an expired token response from the resource server, enters into a refresh flow with the authorization server to obtain a new access token.

Bearer tokens

A *bearer token* refers to any form of identity token that is sent by a client to a resource server using an HTTP authorization header that identifies the authorization scheme as a bearer. The token value of the bearer token authorization header is not defined. However, in most cases, the value is an OAuth2 self-contained access token, or a simple JSON Web Token (JWT).

Example: "Authorization: Bearer abc6324.dgwyuio269nnnwllllsyyyyy.sssy2ajklyuiowl6789"

About JSON Web Tokens

A JSON Web Token (JWT) is commonly used as the physical structure for an OAuth2 token, even though a JWT is not part of the OAuth2 standard. A JWT holds a resource owner's identity, client identity, issuing and expiration timestamps, and scopes used by a resource server's authorization process. Other authorization server-defined claims (also known as JSON fields) can be added by a vendor's implementation. A JWT is a Base64-encoded value that contains three sub-structures that describe the data integrity signature algorithm, the user's identity claims, and the data-integrity signature. Each sub-structure is delimited by a period ('.').

Structure name	Description
Header	Contains information about the JWT payload's and data integrity signature structures
Payload	Contains multiple JWT claims (JSON fields) that describe an authenticated user's ID, issuing and expiration timestamps, scope, client ID, and other values
Signature	Contains a binary digital signature of the header and payload structures, that is produced using the signature algorithm type and size contained in the header

A JWT's payload has a small number of common user identity claim fields, but for the most part is wide open to customization of what it contains. Common identity claim fields include:

JWT field name	Description	Required in JWT token	Required in OAuth2 token

iss	The URI of the authorization server that issued the JWT token.	Yes	Yes
sub	The user ID. For OAuth2, it is the resource owner's ID.	Yes	Yes
aud	Can be a string value holding a single recipient identifier (also known as resource server ID) that limits the token's access to a single resource server. Can also be an array of strings with each array element holding a single recipient identifier, which then allows the token to access multiple resource servers.	Yes	Yes
client_id	This identifies the unique client ID registered by a client application with an authorization server, and is used by a resource server to determine whether the issued JWT can be used to access resource owner data.	No	Yes
jti	A unique identifier for this token that can be used to detect replay attacks and establish client login sessions.	optional	optional
iat	The JWT creation date.	Yes	Yes
exp	The JWT expiration date, after which it can not be used by a resource server.	optional	optional
nbf	The JWT validation date, before which it can not be used by a resource server.	optional	optional

scope	OAuth2 field name that carries a space delimited list of scopes that serve to tell a resource server what data and operations the client is authorized to access.	Yes	Yes
token_type	OAuth2 field name. Indicates the HTTP authorization header scheme that this token was issued in (typically the bearer value).	optional	optional
<other>	Any other authorization server-provided claims.	optional	optional

A JWT's header can specify one of a set of data-integrity algorithm types based on the JWS standard, which includes HMAC (secret-key) and RSA (public-private key) types.

Header field name	**Description**	**Required**
alg	The JWS algorithm name used for generating and verifying the JWT's signature field value. For more information, see the table of JWS signature names.	Yes
typ	The type of token data format. If not specified, it often defaults to JWT by most vendor implementations.	No
kid	The alias key ID name that identifies a keystore entry that contains the encryption key value used by the resource server to validate the JWT's signature field value.	No

WS signature names:

JWS algorithm name	**Cryptographic algorithms**	**Key size**
HS256	HMAC w. SHA	256
HS384	HMAC w. SHA	384
HS512	HMAC w. SHA	512
RS256	RSA signature w. SHA	256
RS384	RSA signature w. SHA	384
RS512	RSA signature w. SHA	512

About support for OAuth2 and JWT in PAS for OpenEdge

PAS for OpenEdge support for OAuth2 allows your ABL business application or data service to act as a resource server that accepts either a OAuth2 self-contained access token (in JWT format) or a simple JWT token. The support for JWT and OAuth2 self-contained access tokens relies on having access to the user's identity information in order to generate a CLIENT-PRINCIPAL object that is usable in your ABL business application for setting the user of an OpenEdge database connection (and therefore tenancy and audit trail). Only those two types of tokens contain sufficient identity information to create a CLIENT-PRINCIPAL object.

Note: The OAuth2 random access token type is not supported because OpenEdge does not embed or interoperate with an external authorization server that is capable of sharing an OAuth2 self-contained access token with PAS for OpenEdge

Main steps for the HTTP request process

As an OAuth2 resource server, PAS for OpenEdge does the following on each HTTP request:

1. Obtains the authorization header's bearer token
2. Validates the bearer token type as either a JWT or OAuth2 self-contained access token (also known as a form of JWT)
3. Validates the JWT's signature using the JWT header's algorithm type and the PAS for OpenEdge configured encryption key
4. Validates the JWT's required payload claims
5. If the bearer token type is a JWT or OAuth2 self-contained access token, validates the required payload claims
6. Authorizes the token's user to the PAS for OpenEdge data services using the granted JWT scope claim
7. Creates an equivalent OpenEdge CLIENT-PRINCIPAL object that is delivered to the ABL business application with each request

PAS for OpenEdge support for OAuth2 is supplied by Spring Security. PAS for OpenEdge extends the core Spring Security OAuth2 project implementation to blend it into the same customer ABL application environment as it does for all other Spring Security authentication and URL data authorization services. That OpenEdge integration includes the formatting of error responses and the creation of CLIENT-PRINCIPAL objects that are passed to the ABL business application.

The OAuth2 and JWT standards offer implementation vendors many design and run-time use-cases that result in many configuration properties. The `oeableSecurity.properties` file found in the OEABL web application's `WEB-INF/` directory contains a common subset of those properties and gives you the ability to configure each OEABL web application independently. The full set of configuration properties is found in the PAS for OpenEdge instance's `conf/oeablSecurity.properties` file, which compose the default values if they are not found in the OEABL web application's configuration. OpenEdge provides many default property values that do not require changing. However, not all properties can contain useful values and must be configured for each installation.

PAS for OpenEdge supports most OAuth2- or JWT-compliant authentication and authorization services. However, there are limitations. An OAuth2 access token or JWT must meet the following criteria to be usable by the OAuth2 support in PAS for OpenEdge:

1. The access token, or JWT, must be received in an authorization HTTP header with the authorization-scheme set to to bearer.
2. The token must contain a user identity (sub) claim to populate the CLIENT-PRINCIPAL User-ID field.
3. The token must contain a scope (scope) claim that can be used by Spring Security to authorize access to the application URLs.

4. The token must contain a resource ID claim that corresponds to a resource ID that is configured in the OEABL web application's security.
5. An OAuth2 access token must contain a client ID claim (`client_id`) that indicates that it was issued to a client type that the OEABL web application is configured to support.
6. The token's signature algorithm must be one of the JWS, HMAC, or RSA types.
7. The encryption key to validate the token's signature field must be available from the issuer of the token and configured.
8. If the token issuer adds an expiration claim (exp), then the token must pass the date-time expiration test.

The following topics provide details about configuring PAS for OpenEdge's OAuth2 support:

- OAuth2 configuration files on page 43
- Enable OAuth2 support on page 44
- Configure a PAS for OpenEdge web application as an OAuth2 Resource Server on page 44
- Configure OAuth2 or JWT Token Validation Services on page 44
- Configure JWT/OAuth2 self-contained access token conversion to an OpenEdge CLIENT-PRINCIPAL on page 50

Note: Do not proceed without first understanding basic OAuth2, JWT, and OpenEdge CLIENT-PRINCIPAL terms and functionality, as explained in the following topics:

- About OAuth2 on page 35
- About OAuth2 security considerations on page 37
- About OAuth2 tokens on page 37
- About JSON Web Tokens on page 39

OAuth2 configuration files

The OAuth2 configuration in PAS for OpenEdge involves these files:

- `oeablSecurity.properties`—Contains property settings for JWT and OAuth2 validation operations and CLIENT-PRINCIPAL object generation.

  Note:

  There is a hierarchy of `oeablSecurity.properties` files, which exist in the following locations:

  - *`oe_install_dir`*`/servers/pasoe/conf/`
  - *`instance-name`*`/conf/`
  - *`instance-name`*`/ablapps/`*`abl-app-name`*`/conf/`
  - *`instance-name/web-app-name`*`/WEB-INF/`

  See Enable ABL application authentication for more information.

- `oeablSecurityJWT.csv`—Contains the resource server's URL access controls for JWT and OAuth2 client access based on the token's claim (claim) field

- `oauth2ResSvcClients.cfg`—A file containing OAuth2 client authorization information for client ID (client_id) field (not required for standard JWT token validation)

Enable OAuth2 support

OAuth2 support has some fundamental differences that set it apart from other direct user login and OpenEdge SSO processes available in PAS for OpenEdge. Therefore, it has its own Spring Security LoginModel configuration that cannot be combined with any type of direct user login.

Note: If multiple forms of client authentication are required for a single OpenEdge ABL application, then a second OpenEdge web application should be deployed and mapped to the same ABL application, where the alternate user login can be configured.

Enabling the OpenEdge web application in PAS for OpenEdge to support OAuth2 support is accomplished by setting the `client.login.model` property in the `oeablSecurity.properties` file to `oauth2`.

Note: When `client.login.model` is set to `oauth2`, the `http.all.authmanager` property value has no effect.

Configure a PAS for OpenEdge web application as an OAuth2 Resource Server

Configuring a PAS for OpenEdge web application as an OAuth2 resource server requires the following:

- Configure OAuth2 or JWT Token Validation Services on page 44
- Configure JWT signature field validation on page 45
- Configure JWT payload field validation on page 47
- Configure self-contained access token validation on page 48
- Authorize access to URLs and HTTP Methods on page 48
- Configure JWT/OAuth2 self-contained access token conversion to an OpenEdge CLIENT-PRINCIPAL on page 50

Configure OAuth2 or JWT Token Validation Services

Set the token services validation type

At the core of Spring Security's OAuth2 support is the validation of an access token. The payload of the access token must minimally contain a set of *claim fields*. That minimal list varies depending upon whether the token is an OAuth2 access token or a simple JWT.

To configure PAS for OpenEdge for the type of token payload validation obtained from the HTTP request's authorization header, set the following property in `oeablSecurity.properties`:

```
oauth2.resSvc.tokenServices={ jwt | oauth2 }
```

`jwt`

`jwt tokenServices` looks for a `jwt` in the HTTP request's HTTP authorization header's *bearer* scheme value. If a JWT-type bearer token is found, then its signature, and required JWT payload

claims (aud and exp), are validated before they are passed to the URL authorization process. Using JWT tokens has more risk because it exposes the token's information to untrusted clients, such as a browser in an internet client. This selection is most useful for cases where intranet clients obtain a JWT from a non-compliant OAuth2 authorization server and use it to access a resource server without the need for full formal OAuth2 validity checking

`oauth2`

`oauth2 tokenServices` incorporates `jwt tokenServices` and adds additional checking for required OAuth2 standard payload claims. This selection is most useful for cases where either internet or intranet clients follow one of the four OAuth2 authorization flows to an authorization server.

Configure OAuth2 sessions

An OAuth2 resource server hosts a stateless REST API for its clients. As a stateless REST API, it does not create user HTTP sessions, but does not preclude the REST API's implementation from creating them. Spring Security provides a property to control the generation and use of HTTP sessions. The default is to follow the normal REST API stateless model.

This can be changed to have Spring Security use and maintain HTTP sessions by setting the following property's value in `oeablSecurity.properties` to false:

```
oauth2.resSvc.stateless={ true | false }
```

Configure the WWW-Authenticate realm name

An OAuth2 resource server accepts access tokens using the HTTP authorization header. If the resource server's validation of the access token fails, then it will return a 401 status and a `WWW-Authenticate` HTTP header with a `realm-challenge` phrase. The response's realm-challenge phrase can be customized by setting the following property value in `oeablSecurity.properties`:

```
oauth2.resSvc.realmName=oeoauth
```

Configure JWT signature field validation

Validating a JWT access token is an essential step in the OAuth2 resource server authentication and authorization processes. All JWT access tokens must pass the payload requirements, data integrity signature, and (optionally) expiration. If any of these checks fails, then the client's HTTP request is rejected.

A JWT's data integrity signature supports multiple types of algorithms. JWT support in PAS for OpenEdge includes HMAC and RSA (public and private) algorithms, as specified in the JSON Web Signature (JWS) standard.

Configure the HMAC or RSA type

Choose the JWT signature validation to be performed by setting the following property in the `oeablSecurity.properties` file:

```
jwtToken.signatureAlg={ HS256 | HS384 | HS512 | RS256 | RS384 | RS512 }
```

Note: All HS*xxx* types apply to the HMAC signature algorithm. All RS*xxx* types apply to the RSA (public and private) signature algorithm.

Configure the HSxxx signature validation encryption key

The HS*xxx* encryption key is a single value configured in the `oeablSecurity.properties` file:

```
jwtToken.macKey=value
```

value

A string value, either clear text or encoded text. Encoded text is the output of *oe-install-dir*`/bin/stspwdutil`

Note: Because the encryption key is a single value, PAS for OpenEdge supports only one authorization server for each OEABL web application configuration.

Configure the RSxxx signature validation encryption key

Support for accessing the RSA public keys includes:

- A Java (`.jks`) encrypted keystore file containing a digital certificate holding the single RSA public key
- A file system directory holding the single PEM-encoded RSA public key
- A JSON Web Key (JWK) set that holds multiple RSA public keys, and is selected by the JWT header's `kid` claim

Configure a .jks Java keystore

Set the following properties in the `oeablSecurity.properties` file:

```
jwtToken.keystore.type=mac
jwtToken.jksKeyStore=path-to-jks-keystore
```

Note: The keystore is maintained by using the Java `keytool` utility. See Oracle documentation for more information.

```
jwtToken.jksKeystore.pwd=value
```

value

A string value, either clear text or encoded text. Encoded text is the output of *oe-install-dir*`/bin/stspwdutil`

```
jwtToken.jksKeystore.alias=sample
```

sample

The name of the keystore entry's alias used when you store the JWT RSA signature's public-key certificate that you obtained from the authorization server's operators.

Configure a PEM-encoded RSA public key

All files holding PEM-encoded RSA public keys must be in the form *file-name*.pem. You must configure the file system directory path (which acts as a keystore) and the alias for finding the PEM file within the file system directory:

```
jwtToken.keystore.type=pem
jwtToken.pemKeyStore=<file-system-path to PEM files>
jwtToken.pemKeystore.alias=<PEM file-name>
```

Note:

- The PEM-encoded file may optionally contain the RSA private key.
- A sample PEM-encoded file can be found in a PAS for OpenEdge instance's conf/jwtkeys directory.

RSA public/private key pairs can be generated for testing by using the following two command lines:

```
oe-install-dir/bin/sslc genpkey x-out filename.prv.pem -outform PEM -algorithm RSApkeyopt
 rsa_keygein_bits:key-bit-size

oe-install-dir/bin/sslc rsa -pubout -in filename.prv.pem -out filename.pub.pem
```

Configure JWK set holding RSA public keys

A JWK set is a JSON structure that holds one or more individual JWK JSON objects. Each JWK object contains a single RSA public key and can be identified by an alias. When a JWT is received and its header contains a kid claim, the kid value is used as the alias that is used to look up a JWK object. If a JWT's header does not contain a kid claim, or a JWK cannot be resolved by alias, the request is rejected.

```
jwtToken.keystore.type=jwk
jwtToken.jwkKeyStoreURL=URL of where JWK RSA public keys will be loaded from
```

Configure JWT payload field validation

The following properties are used to validate a JWT self-contained access token's payload fields.

Validate the resource owner's ID claim

A JWT has a recommended field name (sub) to hold the authenticated user's ID, but can be configured differently by the issuing authorization server. See the JWT issuer's documentation for the field name that contains the authenticated user's ID:

```
jwtToken.usernameField=sub
```

Configure the resource server's ID

Each OAuth2 resource server has a well-known ID that it registers with the authorization server and is used to filter out the JWT tokens issued for other resource servers. The authorization server inserts a payload audience (`aud`) claim, which is verified by the PAS for OpenEdge JWT validation process.

To configure the OAuth2 resource server ID, change the default `oeablapp` value for the one registered with the authorization server:

```
oauth2.resSvc.audience=oeableapp
```

Configure self-contained access token validation

Note the following the properties used to validate the additional claims that are found in a self-contained access token's payload.

An OAuth2 access token's payload must contain a `client_id` claim that uniquely identifies each type of resource server client. You must configure a client ID for PAS for OpenEdge when you have configured the OAuth2 service type to be oauth2:

```
oauth2.resSvc.tokenServices=oauth2
oauth2.resSvc.clientCfg=../oauth2ResSvcClients.cfg
```

The resource server's client configuration is normally found in the OEABL web application `WEB-INF/oauth2ResSvcClients.cfg` file, and has the following XML element format:

```
<oauth:client-details-service id="oauth2ResourceClientDetails">
           <oauth:client client-id="oeablClient" />
               ...
      </oauth:client-details-service>
```

Edit the file and change the `client-id` attribute's value to hold the OAuth2 client's registered ID.

Authorize access to URLs and HTTP Methods

The OAuth2 access tokens do not contain user IDs and roles, in the traditional sense, for authorizing access to resources. Although a user ID exists in a self-contained access token for logging and audit trail use, it is not used outside of that scope. Instead, OAuth2 uses the notion of *client-id* and *scope* to authorize a particular client (application) to use the resource server and to define the operations that a resource owner has authorized the resource server to perform for that client.

The traditional Spring Security Role-Based-Authorization (RBA) and its configuration do not apply to OAuth2. Instead, a new form of authorization access control and configuration is used by Spring Security to grant the right to perform an operation.

Basically, OAuth2 authorization is configured using a file named `WEB-INF/oeablSecurityJWT.csv`. This file performs the same basic function as the traditional RBA configuration file, `oeablSecurity.csv`.

The format of the `oeablSecurityJWT.csv` file has the same three double-quote fields that hold the URI, HTTP-method, and access-control specification. The first two fields (URI and HTTP method) are the same as in the RBA's `oeablSecurity.csv` file. The third field holding the access-control specification uses a different Spring Security expression language.

The expression language used for traditional RBA and OAuth2 scope authorization share the concept of testing whether a security token is granted a *name*, and that *name* corresponds to granting access to some process, function, or data. Whoever grants the *name* and the resource server who associates it with access to some resource agree on what the *name* is.

In the case of OAuth2, the *name* is an OAuth2 *scope* that the authorization server inserts into the self-contained access token as directed by the resource owner. The same rules apply as with RBA: the resource server's configuration must use the *scope* names configured and used by the authorization server. In production environments, that information must be obtained by the administrator.

An OAuth2 access-control expression has the following format:

```
access-control-expr := " oauth2-spec [ { or | and } oauth2-spec ... ] "
```

```
oauth2-spec = #oauth2.test( 'value' [, 'value'... ] )
```

The following table is a list of tests:

`clientHasAnyRole('name'...)`	Checks if OAuth2 client has one of the listed role names
`clientHasRole('name')`	Checks if the OAuth2 client as the specified role name
`denyOAuthClient( )`	Denies access to all OAuth2 clients, but allows other types of clients
`hasAnyScope('name' ...)`	Checks if the OAuth2 client has any named scope specified
`hasAnyScopeMatching( 'reg-ex'...)`	Checks if the OAuth2 client has one of the scopes matching a specified regex expression
`hasScope('name')`	Checks if the OAuth2 client has the named scope specified
`hasScopeMatching('reg-ex')`	Checks if the OAuth2 client has a named scope matching a specified regex expression
`isClient( )`	Checks if the request is from an OAuth2 client (and not another type of user)
`isOAuth( )`	Permits only OAuth2 type requests
`isUser( )`	Checks if the current request is from a user and not from an OAuth2 client

Each test returns true or false, and the entire expression must return true for the resource server to grant access.

Note:

In the default distribution of `oeablSecurityJWT.csv` the default scope names are:

- PSCUser
- PSCAdmin
- PSCDebug

Configure JWT/OAuth2 self-contained access token conversion to an OpenEdge CLIENT-PRINCIPAL

Similar to all Spring Security processes in PAS for OpenEdge, a successful authentication and authorization produces an OpenEdge CLIENT-PRINCIPAL object that is delivered to the ABL application code on each request.

The following table indicates the mapping of self-contained JWT fields to OpenEdge CLIENT-PRINCIPAL attributes:

JWT field	CLIENT-PRINCIPAL attribute
sub (by configuration)	User-id
exp	expires
scope	Roles
<misc> (if configured)	Properties

Configure mapping the JWT's scope claim to Spring and CLIENT-PRINCIPAL roles

The JWT claim field is a comma separated list of values, where each value is mapped to a role name in the Spring Security and CLIENT-PRINCIPAL object tokens. Those mapped role values are used to authorize the client's access to specific URLs:

```
jwtToken.mapScopeToRole={ true | false }
```

A JWT has a recommended field name `scope` to hold the scope of the authenticated user. However, you can configure this field name, as shown below, using the authorization server that issues it:

```
jwtToken.scopeNameField={ scope }
```

If the configured field name for scope is not available in the JWT, then the JWT uses `PSCUser` as the default scope. You can set this default scope using the `jwtToken.defaultRoles` property in the `oeablSecurity.properties` file.

Refer to the JWT issuer's documentation to determine the field name that contains the scope of the authenticated user and map its claim to the scope (role).

Configure JWT claims as CLIENT-PRINCIPAL property value

JWT tokens may contain any number of authorization server defined payload claims. If you want those claims made available to the ABL application using a CLIENT-PRINCIPAL object, then set the following option:

```
jwtToken.includeAllClaims={ true | false }
```

The creation and sealing of a CLIENT-PRINCIPAL object is handled by the `OEClientPrincipalFilter` and follows its usual configuration rules.

Debug logging for OAuth2

OAuth2 support can be troublesome to configure due to the various actor implementations by different vendors. When client access is denied, you may need to troubleshoot your Spring OAuth2 configuration. It is often easier to setup a dedicated PAS for OpenEdge test instance, and enable full Spring Security and OpenEdge logging. It will produce large amounts of information, which can be filtered after you gain a better understanding of the problem area.

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.oauth2" level="DEBUG" />
<logger name="org.springframework.security.jwt" level="DEBUG" />
<logger name="org.springframework" level="DEBUG"/>
<logger name="com.progress.appserv.services.security" level="DEBUG"/>
```

Extend OpenEdge SSO to web applications

OpenEdge supports a single-sign on (SSO) framework in its application server and database product clients. PAS for OpenEdge extends the OpenEdge SSO framework to include support of mobile and browser-based clients (HTTP Clients) for OpenEdge ABL web applications. Providing tight integration with the existing SSO framework, SSO support for HTTP clients enhances ABL web application security, reduces the inconvenience of multiple logins, and relieves the need to write and maintain code that requires a high degree of technical security expertise.

The following topics describe how the existing OpenEdge SSO framework extends to web application clients, how that extended functionality is secured, and how the web application clients use those SSO services.

About PAS for OpenEdge SSO technologies

The PAS for OpenEdge extends OpenEdge SSO functionality into the Spring Security framework authentication and authorization process.

OpenEdge SSO support continues to authenticate user IDs and passwords, and to produce CLIENT-PRINCIPAL tokens for ABL application use. That basic OpenEdge functionality is extended with the capability to issue a simple type of SSO token to a client that it can pass to other PAS for OpenEdge web services that have SSO enabled. However, the token may not be refreshed (that is, its expiration extended) if the originating user login session terminated. If the SSO token is issued without the ability to be refreshed, then forcing a user to re-authenticate, the login session expiration limitation is moot.

The PAS for OpenEdge SSO support extends the original RESTful request support and adds a Spring Security filter to handle an upgraded client-server protocol. The PAS for OpenEdge SSO extensions provide both SSO token Producer and Consumer features for generating and consuming SSO tokens, respectively. Any one PAS for OpenEdge web service can be configured to produce, consume, or both. Additionally, a Spring Security configuration template (`oeablSecurity-oesso.xml`) allows a PAS for OpenEdge web service to only be accessed using a PAS for OpenEdge SSO token.

Because of the Spring Security framework's plug-in Authentication Provider architecture, PAS for OpenEdge SSO token generation can be configured to use any user-account provider supported by Spring Security or by an OpenEdge: properties file, such as LDAP; Active Directory; OERealm; SAML; OpenID; or CAS.

PAS for OpenEdge SSO tokens

The SSO access token in PAS for OpenEdge is a Base64-encoded and sealed CLIENT-PRINCIPAL. An optional Refresh token is a unique string value that is paired to one CLIENT-PRINCIPAL token.

A CLIENT-PRINCIPAL token minimally contains these fields:

- User ID and OpenEdge domain
- State SSO
- Expiration
- Roles (as granted by the Spring Security framework's existing behavior)
- Scope (which supplements Roles as a mechanism to further refine authorization rules. It limits clients with certain access tokens to certain web services, before Role URL authorization is tested.)

OpenEdge domain seal and validation support

PAS for OpenEdge SSO supports the sealing and validation of both single and multi-tenant CLIENT-PRINCIPAL tokens using the `OEClientPrincipalFilter` bean, which is configured in the `oeablSecurity.properties` file. The `OEClientPrincipalFilter` bean becomes the single point that manages all aspects of translating Spring tokens to CLIENT-PRINCIPAL tokens, the sealing of CLIENT-PRINCIPAL tokens, and the validation of CLIENT-PRINCIPAL tokens across all methods of direct-login and SSO.

All OpenEdge products that produce or consume CLIENT-PRINCIPAL tokens, including PAS for OpenEdge, are expected to define and use unique OpenEdge domain names, with each OpenEdge domain having a secret access code that is configured in each product that uses the OpenEdge domain. Each OpenEdge product is configurable to use some or all of the enterprise's defined OpenEdge domains, in effect adding a level of domain authorization to a Progress component. PAS for OpenEdge supports this architectural model

PAS for OpenEdge's web service SSO requires configuring a minimum of one domain or access code pair, and may support multiples by using a multi-domain registry (generated by the GENDOMREG utility). The ABL business application that receives an SSO token is expected to be configured with the same OpenEdge domain names and access codes. An SSO token must be validated using a domain configuration before it can be accepted as proof of the client's ability to use services and to be passed to the ABL business applications.

PAS for OpenEdge SSO configuration guide

Configuring PAS for OE SSO tokens is accomplished by updating the following files:

File path	Description
instance-name/conf/oeablSecurity.properties	Spring Security configuration defaults for all web applications
instance-name/webapps/*web-app-name*/WEB-INF/oeablSecurity.properties	Spring Security configuration settings for an individual web application
instance-name/webapps/*web-app-name*/WEB-INF/oeablSecurity.csv	URL access controls (Spring Security intercept-url settings) for individual web applications

Note: The `oeablSecurity.properties` files are where you configure the `OEClientPrincipalFilter` bean which manages all aspects of translating Spring tokens to CLIENT-PRINCIPAL tokens, the sealing of CLIENT-PRINCIPAL tokens, and the validation of CLIENT-PRINCIPAL tokens across all methods of direct login and SSO.

There are two SSO configurations, one for web applications that produce SSO tokens and one for web applications that consume SSO tokens.

Table 2: Overview of SSO Producer Configuration

Configure CLIENT-PRINCIPAL creation	• Add single or multi domain and access codes • Include Spring Security Authentication Provider granted roles • (Optional) Static Spring Security Roles for authorization to URLs

Configure SSO token creation	• Enable SSO token creation
	• Optionally change initial expiration time from 3600 seconds
	• Optionally enable SSO Token Refresh operations
	  • Optionally change refresh delta time of 3600 seconds
	  • Optionally define a SSO Token *scope* to filter which PAS for OpenEdge services are allowed to accept a SSO token generated by this service
	  • Optionally configure error level detail returned to the client
	  • Optionally allow HTTP messages instead of the required HTTPS

Note: Because of the security risks, PAS for OpenEdge web applications should not produce SSO tokens unless there are deployed clients capable of using the SSO that is produced. Therefore, the default setting for authentication and generation of native OpenEdge SSO tokens is disabled. In most cases, you can simply enable authentication or generation, or both.

Table 3: Overview of SSO Consumer Configuration

Configure CLIENT-PRINCIPAL validation	Add single or multi domain and access codes
Configure SSO Token use and validation	• Enable accepting SSO tokens for access to service URLs • Optionally configure error level detail returned to the client • Optionally allow HTTP messages instead of the required HTTPS

Configure the validation and use of native OpenEdge SSO token in client requests

The following table is a list of properties in `oeablSecurity.properties` that control if and how OpenEdge SSO tokens can be used to gain access to the data services.

Property name	Data type	Default	Value range	Description
OESSO.error.detail	integer	0	0 (none) 1 (terse) 2 (debug)	Controls the amount of error detail returned to a client for all SSO operations. The default (0) meets security best practices in returning little of value a hacker can make use of. But it does not supply an administrator or end-user with useful information for problem solving. Higher levels provide more information to administrators for problem resolution, but can also provide information usable by a hacker to attack your application. This property is used to set OESSOFilter.authErrorDetail
OESSO.require.https	boolean	true	true|false	When true controls the requirement for all SSO operations to require a client request to be made using the HTTPS URL scheme. When set to false allows HTTP (not recommended for operating a secure web application). This property is used to set the property OESSOFilter.authSecurity
OESSOFilter.authPolicy	string	disabled	—	See the following section, *OE SSO Token Consumer Policies*.
OESSOFilter.authmanager	string	string	—	Control which Authentication Manager is used to validate the OECP SSO token passed by the HTTP client. This property is mapped to the http.all.authmanager property and should only be changed when the OECP SSO Authentication-Manager must be different than the one used in the OECP SSO provider. See the http.all.authmanager property for valid settings.

Property name	Data type	Default	Value range	Description
`OESSOFilter.authScheme`	string	OECP	valid string	The HTTP Authorization header's authentication scheme field name that identifies the presence of an OECP SSO token value **Caution:** OpenEdge recommends that you *do not* change this value. Format: `Authorization: auth-scheme base64-sso-token-value`
`OESSOFilter.authClientType`	string	*	`regex`	Adds the ability to require the HTTP request's User-Agent: header to contain a specific value as defined by a Java RegEx pattern. The default "*" value disables User-Agent: header checking
`OESSOFilter.authErrorDetail`	int	---	---	Mapped to the property `OESSO.error.detail`. Explicitly defines a value if the error detail for consuming SSO tokens should be different from other SSO operations.
`OESSOFilter.authSecurity`	boolean	---	---	Mapped to the property `OESSO.require.https` Explicitly define a value if the requirement to use HTTPS should be different from the other SSO operations.
`OESSOTokenManager.ssoAllowScope`	string	*blank*	blank or list	When non-blank this property is used to control which OpenEdge SSO tokens may be used by this web application. If the OpenEdge SSO token passed by a client does has not been granted one of the scope values in this list, the token is rejected and the request fails. See the OpenEdge SSO token provider property `OESSOTokenManager.ssoGrantScope` for additional information.

OpenEdge SSO Token Consumer Policies

Policy Name	Description
`disabled`	The web application will not look for, or handle OpenEdge SSO tokens. If a native SSO token is passed in an Authorization header by a client it will be ignored

Policy Name	Description
`sufficient`	The web application will look for an HTTP Authorization header containing an authentication-scheme specified by the `OESSOFilter.authScheme` property (OECP) and a native SSO token. • If no HTTP Authorization header is present the authentication process continues with the next configured filter, which allows other SSO filters to operate. • If an HTTP Authorization header is present, but does not contain an authentication-scheme field matching the `OESSOFilter.authScheme` property, the authentication process continues with the next configured filter, which allows other SSO filters to operate. • If an HTTP Authorization header is present,and contains an authentication-scheme field matching the `OESSOFilter.authScheme` property, it will validate the contained OpenEdge SSO token value. If a validation error occurs, a 401 error will be returned to the client and no other SSO filters will be invoked. If validation is successful the native token will be extracted, the remaining authentication filters will be skipped, and the native token's ROLEs will be used to perform URL authorization. This policy is best used in the OpenEdge SSO token producer and OpenEdge SSO token consumer web applications that also support other forms of Spring Security direct-logins to user accounts.
`required`	

Policy Name	Description
	The web application will look for an HTTP Authorization header containing an authentication-scheme specified by the `OESSOFilter.authScheme` property (OECP) and a native SSO token. A 401 error response will be returned to the client and no other SSO filters will be invoked if any of the following conditions fail: • No Authorization header is found • An Authorization header is found, but is blank • An Authorization header is found but its authentication-scheme does not match the OESSOFilter.authScheme property • An Authorization header is found, has an authentication-scheme that does not match the `OESSOFilter.authScheme` property • An Authorization header is found, has an authentication-scheme that does match the `OESSOFilter.authScheme` property but does not have a SSO token value • An Authorization header is found, has an authentication-scheme that does match the `OESSOFilter.authScheme` property but does not have a valid SSO token value If successful the native token will be extracted, the remaining authentication filters will be skipped, and the native token's ROLEs will be used to perform URL authorization. This policy is best used in the OpenEdge SSO token consumer web applications that do not support any other forms of Spring Security direct-login to user accounts.

Configure the generation of OpenEdge native SSO tokens

Only certain types of PAS for OpenEdge web application may be a source of native SSO tokens. External authentication systems that produce their own security tokens and are integrated into Spring Security's process are prohibited from having an SSO token generated by OpenEdge.

The generation of an OpenEdge native SSO token occurs after Spring Security's authentication process completes a successful user direct-login. This post processing operation generates an extended ClientPrincipal security token that is safe enough to function in a less secure client login context environment (where a client receives and exposes a security token to the network world). This processing is handled by the existing OpenEdge authentication success handling, which uses the OpenEdge Client-Principal handler to do the physical Client-Principal generation and sealing. The OpenEdge successful authentication handler has the following configuration properties for producing OpenEdge SSO tokens, and adds some optional properties to allow tailoring to individual installations:

The following table is a list of properties in `oeablSecurity.properties` that control OpenEdge token generation properties.

Property Name	Data Type	Default	Value Range	Description
`OESSO.require.https`	boolean	`true`	`true` | `false`	When `true`, controls the requirement for all SSO operations to require a client request to be made using the HTTPS URL scheme. Set to `false` to allow HTTP (not recommended for operating a secure web application). This property is used to set the `OEAuthnSuccessHandler.tokenSecure` property.
`OESSO.error.detail`	integer	`0`	0 (none) 1 (terse) 2 (debug)	Controls the amount of error detail returned to a client for all SSO operations. The default (0) meets security best practices in returning little of value a hacker can make use of. But it does not supply an administrator or end-user with useful information for problem solving. Higher levels provide more information to administrators for problem resolution, but can also provide information usable by a hacker to attack your application. This property is used to set the `OEAuthnSuccessHandler.tokenErrorDetail` property.
`OESSOTokenManager.tokenPolicy`	string	disabled	—	Controls the actions to take when producing and returning a native OpenEdge SSO Token to a client. See the following *OESSOTokenManager Policies* section for more information.

Property Name	Data Type	Default	Value Range	Description
`OESSOTokenManager.ssoTokenURLOption`	string	oesso	`“oesso”` `“”`	The URL query option name used by a client to request the server issue a OpenEdge SSO token if the `OESSOTokenManager.tokenPolicy` value is `ifRequired`. For example: https://yatserv/static/ubj_pingsecurity_ok?tokenURLOption=yes|no A `yes` option value requests a OpenEdge SSO token be created and returned to the client if the user's authentication is successful. Any other option value will not create or return an OpenEdge SSO token. Ignored if the OESSOTokenManager.`tokenPolicy` property value is `disabled` or `always`.
`OESSOTokenManager.ssoTokenRefresh`	boolean	`true`	`true|false`	Controls a client's ability to request that an expired OpenEdge SSSO be refreshed with a new expiration date. NOTE: This property's value is forced to `false` if the `OESSOTokenManager.tokenPolicy` property's value is `disabled`.
`OESSOTokenManabger.ssoTokenExpires`	integer	`3600` (1 hour)	`0` – *n* seconds	This property controls the number of seconds a newly created OpenEdge SSO token is valid before it expires. If the value is less than 1, no refresh token will be generated and returned
`OESSOTokenManager.ssoGrantScope`	string	`""`	string	This property can be used to control which OpenEdge web applications may use an OpenEdge SSO Token produced by the Token Manager. It is used when an ABL application has many web applications, but not all of them should accept any OpenEdge SSO token. The *`string`* a comma separated list of names that correspond to web applications that should accept the OpenEdge SSO token. When blank, no scope information will be included in the OpenEdge SSO token. When Refer to the OpenEdge SSO token consumer property `OESSOTokenManager.ssoAllowScope`.

Property Name	Data Type	Default	Value Range	Description
OEAuthnSuccessHandler.tokenErrorDetail	integer	0	0-3	Controls the amount of error detail returned to a client during the OpenEdge SSO token creation process. This property is normally set using the OESSO.error.detail property to keep error detail information level consistent across the different SSO process operations. The property's value may be explicitly set if the OpenEdge SSO token generation errors return a different amount of error information than other SSO operations.
OEAuthnSuccessHandler.tokenSecure	boolean	true	true|false	Controls the requirement for HTTPS requests from the client while authenticating the user and returning an OpenEdge SSO token. This property is normally set using the OESSO.require.https property to keep the requirements for using HTTPS consistent across the different SSO process operations.This property may be explicitly set independently if the requirement for HTTPS is different from other SSO operations.

OESSOTokenManager Policies

Policy Name	Description
disabled	The web application will not generate OpenEdge SSO tokens. This policy must be used in all web applications that do not support the Spring Security HTTP Form direct-login to user accounts.

Policy Name	Description
`ifRequired`	The web application will only produce an OpenEdge SSO token if the client application requests it by using adding a URL option during a successful direct-login operation. For more information refer to the `OESSOTokenManager.tokenURLOption` property This policy should be used only in OpenEdge SSO token producer web applications that support the Spring Security HTTP Form type of direct-login to user accounts
`always`	The web application will always produce a OpenEdge SSO token if the client completes a successful direct-login operation. This policy should be used only in OpenEdge SSO token producer web applications that support the Spring Security HTTP Form type of direct-login to user account. **Note:** This is an extremely dangerous policy setting and should only be used in cases where the web application's authentication process is protected by other controls such as client types, IP addresses, etc.

Configure refresh of OpenEdge native SSO tokens

To mitigate client-side security holes, poor client application code security, and man-in-the-middle attacks, it is essential to limit the window of time that the SSO tokens handled by clients can be used. Using small windows of time requires the ability to transparently refresh an SSO token with a new expiration time. The refresh of an SSO token requires that only the authenticated user has the knowledge that the SSO token shared with the SSO token issuer. This ensures that the SSO token issuer can validate that only the authenticated user can obtain a refreshed SSO token.

The OE SSO token refresh provides a client with the ability to refresh an expired OpenEdge Native SSO token. The implementation intercepts a web application relative-URI, validates the client's refresh token, and, if successful, re-issues a new OpenEdge Native SSO token with a given lifetime.

OE SSO Token Refresh Properties

Property Name	Data Type	Default	Value Range	Description
OESSO.require.https	boolean	true	true | false	true sets the requirement for all SSO operations to require a client request to be made by using the HTTPS URL scheme. false allows HTTP (not recommended for operating a secure web application). This property is used to set the property OESSORefreshFilter.refreshSecure
OESSO.error.detail	integer	1	0 (none) 1 (terse) 2 (debug)	Controls the amount of error detail returned to a client for all SSO operations. The default (0) meets security best practices in returning little of value a hacker can make use of. But it does not supply an administrator or end-user with useful information for problem solving. Higher levels provide more information to administrators for problem resolution, but can also provide information usable by a hacker to attack your application.
OESSORefreshFilter.refreshURL	string	/static/oe	viable relative URI	Sets the relative URI that a client will use to execute a refresh operation for an expired OE SSO token. The property's value may not be blank or in any of the following URI spaces: • /apsv • /soap • /rest • /web

Property Name	Data Type	Default	Value Range	Description
`OESSORefreshFilter.refreshURLOption`	string	`refresh`	`{ refresh }`	The URL query option name that contains the type of token operation requested by the client. For example: `https://myhost.com/rest/a/auth/token?refresh`
`OESSORefreshFilter.RefreshClientType`	string	`*` (every client type)	Java RegEx pattern	Adds the ability to require the HTTP request's User-Agent: header to contain a specific value as defined by a Java RegEx pattern. The default `"*"` value disables `User-Agent: header` checking.
`OESSOTokenManager.ssoRefreshDeltaTime`	integer	`3600` (1 hour)	`1` – *n* seconds	Controls the amount of time, in seconds, that a refreshed OE SSO token will remain valid before it expires and must be refreshed again.

Property Name	Data Type	Default	Value Range	Description
`OESSORefreshFilter.refreshErrorDetail`	integer	`0`	`0-2`	Controls the amount of error detail returned to a client during the OE SSO token refresh process. This property is normally set using the `OESSO.error.detail` property to keep error detail information level consistent across different SSO process operations. The property's value may be explicitly set if the OE SSO token refresh errors should return a different amount of error information than other SSO operations.
OESSORefreshFilter.refreshSecure	boolean	`true`	`true|false`	Used to control the requirement for HTTPS requests from the client while refreshing an OE SSO token. This property is normally set using the `OESSO.require.https` property to keep the requirements for using HTTPS consistent across the different SSO process operations.This property may be explicitly set independently if the requirement for HTTPS is different from other SSO operations.

PAS for OpenEdge guide to SSO token handling

This topic describes SSO authentication models and is for HTTP client application developers who want to use native OpenEdge SSO token functionality. This topic also contains information about the formation of HTTP requests and interpretation of HTTP responses.

HTTP FORM authentication model

The HTTP FORM authentication model provides user session support when the client uses application defined URL resources for login/logout operations. The client supplies the user's identification assertions (such as user ID and password) in a POST request's body and receives session information in the POST response's headers. For each subsequent HTTP request, the session information received as part of the login operation is passed as HTTP header information.

Applying SSO to this authentication model involves obtaining the native OpenEdge SSO token created and stored as part of the user login process, and passing that SSO token to other web applications that are configured to accept it.

Client request

```
POST web-app-url/static/auth/j_spring_security_check?OECP=yes
Content-Type: application/x-www-form-urlencoded
j_username=userid&j_password=pwd
```

Server response

```
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
set-cookie : JSESSIONID=user-session-reference

{ “token_type” : “oecp”,
 “access_token” : “b64-oecp-sso-token”,
 “refresh_token” : “oecp-ref-token”,
 “expires_in” : int-seconds
}
```

Note: `refresh_token` may be blank, indicating that the `access_token` cannot be refreshed.

HTTP status codes

- **200** indicates successful server response.
- **400** indicates an SSO token generation failure.

  ```
  Content-Type: application/json;charset=UTF-8
  Cache-Control: no-store
  Pragma: no-cache

  “error_description” : “error-desc
  }

   token-error-code
  ```

  For more information about *token-error-code*, see SSO token error codes on page 69.

- **401** indicates user authentication failure for data services.

  ```
  WWW-Authenticate : http-form realm info
  ```

- **500** indicates an internal server failure.

HTTP BASIC authentication model

The HTTP BASIC authentication model does not support user login sessions that can be used for maintaining the server context required for issuing or maintaining SSO tokens and their refresh tokens. Therefore, HTTP BASIC cannot produce SSO tokens.

SSO authentication model

The SSO authentication model for native OpenEdge SSO tokens follows the HTTP BASIC model, where a client's identity is authenticated on each HTTP request. The difference between the two is that HTTP BASIC sends user ID and password authentication on each request, and SSO sends an encoded SSO token in each request.

Client request

The HTTP 1.1 format for the request is:

```
"Authorization : " + token_type + " " + access_token
```

For OpenEdge, the client request is:

```
Authorization : oecp b64-oecp-sso-token
```

HTTP status codes

- **200** indicates success.
- **401** indicates an SSO authentication failure.

  ```
  WWW-Authenticate : oecp error=401,
  error_description=error-desc
  token-error-code
  ```

  For more information on *token-error-code*, see SSO token error codes on page 69.

SSO refresh model

All SSO tokens, including the native OpenEdge SSO token, must expire after some set interval. They can be refreshed either by:

- Performing a full direct login by the client
- Returning to the point where an SSO token was issued and requesting a new token with an extended expiration

Refreshing a native OpenEdge SSO token takes place using a defined URL (similar to the way HTTP form login is implemented). In this case, the client uses a POST request to a URL, and passes the refresh token it received with the last SSO token it obtained. If the refresh operation is successful, it returns a new SSO token and, optionally, a new refresh token.

Client request

```
POST web-app-url/static/auth/token?op=refresh
{ “token_type” : “oecp”,
 “refresh_token” : “oecp-ref-token”
}
```

Server response

```
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
 “token_type” : “oecp”,
 “access_token” : “b64-oecp-sso-token”,
 “refresh_token” : “oecp-ref-token”,
 “expires_in” : int-seconds
}
```

HTTP status codes

- **200** indicates successful server response.
- **401** indicates SSO token generation failure.

```
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
“error” : “401”,
“error_description” : “error-desc”
}

 token-error-code
```

For more information about *token-error-code*, see SSO token error codes on page 69.

SSO token error codes

The following table describes the SSO token-operation error codes.

Error Code	Description
internal_error	An internal error occurred while creating or signing a CLIENT-PRINCIPAL at user authentication time; no SSO token was produced.
expired_token	The client-submitted token expired, and access to the web application was denied.
rejected_token	The token value was rejected.

Error Code	Description
invalid_client	The token request was issued from an invalid client type.
invalid_token	The client-submitted SSO token failed validation, and access to the web application was denied.
corrupt_token	The token submitted to an SSO token is corrupt (could not validate seal), and access to the web application was denied.
invalid_request	The request containing information about the SSO token is incorrect or incomplete, and access to the web application was denied.
unsupported_token	The request contains an unsupported token type, and access to the web application was denied.

Progress Developer Studio for OpenEdge

In OpenEdge Release 11.7.2 Service Pack, Progress Developer Studio for OpenEdge contains the following enhancements:

- **Performance improvements**—You will experience improved performance while working with large projects. You experience the improvements mainly during Initialize OpenEdge tooling and content assistance. The improvements can also be experienced during import of projects, browsing ABL files, propath modifications and switching between the cache settings.
- **Content assist enhancements**—Content assist is improved to include listing parameter names along with the code section names (procedures, functions, methods, etc.) in both the Outline view and content assistance. Listing parameter names helps users as parameter names can carry semantic meaning that aid in understanding the parameter. Also included in this enhancement is moving to the next parameter with a tab press.
- **Readme.txt file**—When you create a project to be deployed to PAS for OpenEdge using REST or WEB transport, you find a `Readme.txt` file in the Project Explorer under the project folder. This file contains short description of files and folders that are displayed under the project folder.
- **Viewing agent log file in Log Viewer**—Select a PAS for OpenEdge server instance in the Server's view and right-click and select **View Agent Log in Log Viewer** to view the agent log file in Log Viewer.
- **Enable Kendo UI Builder automatic build**—Select **Enable Kendo UI Builder automatic build** on the **Build** tab of the **Build properties** page to trigger an automatic build of Kendo UI Builder whenever the content in the Kendo UI Builder folder in the Web UI project changes.

Server Technologies

This section describes the update for Server Technologies that is included in the Release 11.7.3 Service Pack.

Working with OpenEdge JMS Adapter

After you have installed the OpenEdge JMS Adapter during the OpenEdge installation, start the OpenEdge JMS Adapter using the oemessaging scripts located in *openedge-installation-dir*/bin:

- oemessaging.bat (for Windows)
- oemessaging (for Unix)

Note: You do not need an AdminServer to start the OpenEdge JMS Adapter. This is especially useful when you have a PAS for OpenEdge installation running in a DMZ location where an AdminServer is not allowed, and you want to start the OpenEdge JMS Adapter.

Start the Progress OpenEdge JMS Adapter

To start the Progress OpenEdge JMS Adapter, follow these steps:

1. Using proenv, set the JMSPROVIDER environment variable. For example:

```
proenv>set JMSPROVIDER=ProviderName
```

   where *ProviderName* is the JMS provider that is specified in the `-DjmsProvider` parameter in `JavaTools.properties`.

2. Using proenv, Set the JMSCLIENTJAR environment variable. For example:

```
proenv>set JMSCLIENTJAR=JMSCLIENTJAR_path
```

   where *JMSCLIENTJAR_path* is the full path of the JMS client JAR file. The default is `$DLC/sonic/MQ10.0/lib/sonic_client.jar`. If there are more than one jar file, provide the paths as semi-colon separated values.

3. Start the Progress OpenEdge JMS Adapter broker using the `oemessaging start` command. For example:

```
proenv>oemessaging start
```

4. After you have started the OpenEdge JMS Adapter, all available administrative options with their short description are listed in the proenv window. The options are as follows:

   - **S** — Summary of the JMS Adapter broker
   - **D** — Server Details
   - **X** — Add new agents
   - **T** — Trim free agents
   - **K** — Kill by server ID
   - **E** — Stop and Exit the broker gracefully
   - **A** — Abort
   - **L** — Connection Summary
   - **C** — Connection detail for one or more connection handles
   - **Y** — List JMS Adapter properties
   - **Z** — List broker Property value based on property name

OpenEdge JMS Adapter commands

You can run the following commands with the oemessaging scripts from proenv:

- **`oemessaging.bat help`** — Provides the usage details.
- **`oemessaging.bat start`** ***`brokerName`*** — Starts the OpenEdge JMS Adapter broker with the *brokerName* specified. If no *brokerName* is specified, the default broker sonicMQ1 is used.

After you have started the OpenEdge JMS Adapter, all available administrative options with their short description are listed in the proenv window. The options are as follows:

- **S** — Summary of the JMS Adapter broker
- **D** — Server Details
- **X** — Add new agents
- **T** — Trim free agents
- **K** — Kill by server ID
- **E** — Stop and Exit the broker gracefully
- **A** — Abort
- **L** — Connection Summary
- **C** — Connection detail for one or more connection handles
- **Y** — List JMS Adapter properties
- **Z** — List broker Property value based on property name