Progress®



OpenEdge® Getting Started:
OpenEdge Authentication Gateway Guide

OpenEdge®

Table of Contents

Copyright....7

Preface....9

Purpose....9
Audience....10
Using ABL documentation....10
References to ABL compiler and run-time features....10
References to ABL data types....10
Typographical conventions....11
Examples of syntax descriptions....12
Long syntax descriptions split across lines....13
Complex syntax descriptions with both required and optional elements....14
Example procedures....14
OpenEdge messages....15
Obtaining more information about OpenEdge messages....15

Chapter 1: Overview....17

Enhanced OpenEdge Database Connection Security....17
Functional overview....18
Server Management....20
Glossary of Terms....20

Chapter 2: Configuring the OpenEdge Authentication Gateway....23

Configuration overview....24
Configuring domains....25
Configuring domain access codes....27
Domain configuration requirements for Native Token Exchange (SSO)....28
Configuring LDAP Connections in a Security Token Service (STS)....29
Configuring policies....31
Configuring event callbacks....33
Spring Security Configuration....34
Create and deploy a certificate for HTTPS with the OpenEdge Authentication Gateway Server....35
STS server key configuration....36

Chapter 3: Configuring your OpenEdge Database to use the OpenEdge Authentication Gateway....41

Configuring your OpenEdge database....41

SSO Token Exchange....43
STS Client Key (for database server)....44
Start OpenEdge database server 45

Chapter 4: Reference entries....47

ABL reference....47
SET-DB-CLIENT function (enhanced for STS)....48
SETUSERID function (enhanced for STS)....52
AUDIT-EVENT-CONTEXT attribute (enhanced for STS)....55
LOGIN-STATE attribute (enhanced for STS)....55
SEAL( ) method (enhanced for STS)....58
VALIDATE-SEAL( ) method (enhanced for STS)....60
Progress.Security.PAMStatus class (enhanced for STS)....61
AccessDenied property....62
AccessRevoked property....62
AccountDisabled property....62
AccountExpired property....63
AuthenticationAccess property....63
AuthenticationFailed property....63
Custom property....63
InvalidConfiguration property....63
LoginDenied property....64
LoginLockout property....64
MaxTries property....64
MissingCredentials property....64
PasswordExpired property....64
PermissionDenied property....65
Success property....65
UnknownUser property....65
Startup parameters....65
Certificate store path (-certstorepath)....65
Domain (-domain) (for STS capability)....66
Key store path (-keystorepath)....66
No Host Verify (-nohostverify) (for STS capability)....67
No Session Reuse (-nosessionreuse) (for STS capability)....67
OS User (-OSUser) (for STS capability)....68
Security cache size (-secsize)....69
STS logging level (-stsloggінglevel)....70
Database utilities....70
PROUTIL DISABLEAUTHGATEWAY qualifier....71
PROUTIL ENABLEAUTHGATEWAY qualifier....71
Auditing events for OpenEdge Authentication Gateway....72
OpenEdge Authentication Gateway utilities....73
Change the keystore password utility (changeP12pwd)....73

Data export utility....74
STS Client Utility (stsclientutil)....75
STS Key Utility (stskeyutil)....78
STS Connection Role Utility (stsconnroleutil)....83
STS URL Utility....91
Strong Password Encryption Utility (stspwdutil)....94

Chapter 5: ABL constructs that support the OpenEdge Authentication Gateway....97

Copyright

© **2019 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.**

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

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Deliver More Than Expected, Icenium, Kendo UI, Kinvey, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), Sitefinity, 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 Autonomous REST Connector, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustAssembly, JustDecompile, JustMock, Kinvey Chat, NativeChat, NativeScript Sidekick, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, 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.

February 2019

Last updated with new content: Release 12.0

Preface

For details, see the following topics:

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

Purpose

This is a guide to administering the OpenEdge® Authentication Gateway, including database configuration, and using associated ABL elements.

Note: The optional OpenEdge Authentication Gateway is a licensed product that can be separately purchased from Progress.

Audience

This manual is intended for Progress Application Server (PAS) for OpenEdge administrators or ABL developers who need to create, configure, or administer instances of the OpenEdge Authentication Gateway.

Using ABL documentation

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

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

References to ABL compiler and run-time features

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

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

References to ABL data types

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

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

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

Typographical conventions

This documentation uses the following typographical and syntax conventions:

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

Convention	Description
|	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*

This directory . . .	Contains examples for the following documents . . .
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.

Obtaining 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

Overview

This chapter is a brief overview of enhanced database connection security features available in OpenEdge. It focuses on the key component, the OpenEdge Authentication Gateway.

Note: The optional OpenEdge Authentication Gateway is a licensed product that can be separately purchased from Progress.

For details, see the following topics:

- Enhanced OpenEdge Database Connection Security
- Functional overview
- Server Management
- Glossary of Terms

Enhanced OpenEdge Database Connection Security

The OpenEdge Database includes support for enhanced Client connection security beginning in the OpenEdge 11.6.2 release through a network of interdependent OpenEdge product security features. This enhanced database connection security puts the Database Administration (DBA) in control of which OpenEdge clients may establish and use those connections in order to access their server's data. With this new level of connection security the DBA is able to:

- Have broader access to user accounts and authentication services

- Share user authentication processes across multiple OpenEdge databases
- Use Role Based Access to control the per-user ability to connect to their database server and/or change the user of that connection
- Selectively control which OpenEdge installations on the network may utilize its authentication service to establish connections
- Use additional audit trail events to track both successful and unsuccessful database connections with additional information about who, what, and from where the connection request originated

The enhanced database connection security provides server-side control over essential authentication, authorization, and auditing security processes. Because the database server now controls those security processes, there is a higher degree of confidence in who may establish and be permitted (or not) to use a connection and access the database data. Once an OpenEdge database server is configured to use enhanced connection security, all database clients are required to delegate ALL connection-related user authentication, authorization, and connection auditing to the database server, and the database server then also employs an additional level of security in its network connections.

The OpenEdge database attains this enhanced level of connection security through its close relationship with its ABL (and as a future roadmap item OpenEdge SQL) clients and the use of a network authentication service that may be shared with all OpenEdge databases and its clients.

An OpenEdge database server optionally delegates its user authentication process to an OpenEdge Authentication Gateway that bridges the gap between OpenEdge and [strong] user authentication products such as Lightweight Directory Access Protocol

(LDAP), Active Directory (AD), and others. The OpenEdge Authentication Gateway employs the same Domain architecture used in all OpenEdge version 11 releases in order to produce native OpenEdge security tokens, ABL Client-Principals. The native OpenEdge security tokens are used in the database server's connection authorization as well as by the ABL application. The support for ABL client Single Sign-On (SSO) when changing an existing connection's user identity continues to use the same Domain configuration as is found in all OpenEdge 11 releases.

An OpenEdge database server can now directly execute connection authorization using Role-Based Access Controls (RBAC) in the same fashion as it does with auditing. A new OpenEdge database connection role has been added that gives the DBA direct per-user control over which authenticated users (via a sealed Client-Principal identity) are permitted to establish a connection, or to take control of an existing connection. This authorization process is based on the same native OpenEdge security tokens used in all OpenEdge 11 releases. So while the OpenEdge database server optionally employs an OpenEdge Authentication Gateway for all new user authentication, it can also accept sealed Client-Principal tokens from Progress Application Server (PAS) for OpenEdge servers.

The enhanced OpenEdge database connection security also increases the amount of database connection-related information available in its audit trail. Added is the ability to explicitly track failed connection requests and provide an enhanced level of detail including who, where, and what originated the database server connection request.

Functional overview

The OpenEdge Authentication Gateway and related OpenEdge features were designed to ensure secure access to ABL application connections to OpenEdge components. Secured access to OpenEdge components begins with, and is dependent upon, a strong user authentication process that enforces a consistent policy across all of an ABL application's distributed components. A prime example of employing consistent user authentication is an ABL application that uses multiple databases that are accessed by combinations of ABL clients and application servers that span the enterprise's network.

The OpenEdge Authentication Gateway is a key component of a centralized authentication and authorization service for database connections. It is an implementation of a Security Token Service (STS) and is supported by other OpenEdge components, including:

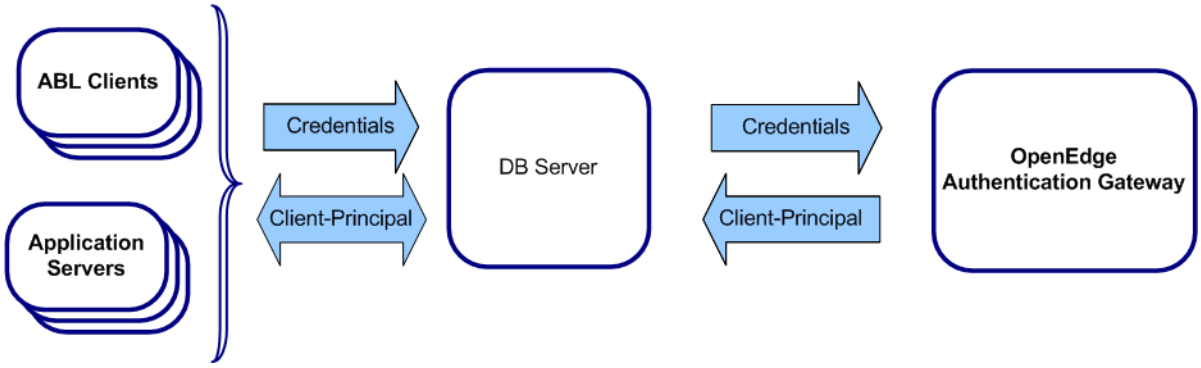
- Utilities to configure OpenEdge databases to access OpenEdge Authentication Gateway services
- Tools to maintain and monitor OpenEdge database activity when using the Authentication Gateway
- ABL functions, handle attributes and methods, and class properties that support OpenEdge Authentication Gateway services

The previous OpenEdge authentication process model allowed for the distribution of security processes among ABL clients and did not include authentication for database connections. With the OpenEdge Authentication Gateway and related OpenEdge features, the sole authentication point is an OpenEdge implementation of an STS. The OpenEdge Authentication Gateway:

- Authorizes access to an STS
- Supports local OS login credentials
- Validates user ID and password credentials
- Creates and passes back a sealed OpenEdge client-principal object (which is an ABL security token)

The Authentication Gateway is a secured Progress Application Server for OpenEdge (PAS) instance where the OESTS web application (`oests.war`) is deployed. The Authentication Gateway requires domain configuration and access to an authentication provider (OS Local, LDAP, Active Directory, etc.) to be able to create and seal a client-principal token.

The architecture addresses authentication and authorization process interactions between an OpenEdge database, its clients, and an OpenEdge Authentication Gateway server. The following diagram is a simplified view of the architecture:



The diagram shows that a client transmits credentials to a database server, which then passes those credentials to an OpenEdge Authentication Gateway. The Authentication Gateway is a centralized authentication point for all clients that request a connection to a particular database. If the credentials are valid, the Authentication Gateway creates a client-principal object, which is validated again before a client connection to the database is allowed.

In addition, note these more detailed aspects of the architecture:

- Connection authorization and connection auditing execute inside the OpenEdge database.
- The OpenEdge database delegates all user authentication (a.k.a., direct login) to the Authentication Gateway.
- The OpenEdge database requires sealed client-principal objects from all clients as proof of user identity in connection authorization and change-user operations.
- The OpenEdge database executes Role-Based-Authorization, employing the user's sealed client-principal object, to grant/deny the user access to the database connection

Server Management

Since the OpenEdge Authentication Gateway is a PAS for OpenEdge instance, you can use the TCMAN command line utility to manage the Authentication Gateway. TCMAN actions include (but are not limited to):

- Starting and stopping the Authentication Gateway
- Displaying information about the configuration and environment
- Managing configuration properties
- Showing the running status of the Authentication Gateway
- Managing server log files

For more information on TCMAN actions and how to use them, see the *TCMAN Reference* in *Progress Application Server for OpenEdge: Administration Guide*.

Glossary of Terms

The following is a list of definitions of the terms used in this document.

Auditing	The secure recoding of security-related events across an application, its utilities, and/or its databases
Authenticate	To confirm an individuals assertion of identity as a precursor to issuing them a *security token*
Authentication	The process of verifying the identity of a user and issuing them a security token as a precursor to authorizing their access to protected resources
Authentication system	In OpenEdge an authentication system is a configurable Domain plug-in that is used by its authentication processes to authenticate a user's identity as a member of that Domain
Authentication Provider	The Spring Security framework's equivalent to an OpenEdge Domain's authentication system plug-in
Authorize	Grant/deny access to a protected resource
Authorization (process)	The process of using a user's security token to Authorize their access to some protected resource
Client-side security	The execution of authentication, authorization and auditing processes by application/database clients
Client-Principal	An ABL language handle used to access the contents of an OpenEdge security token

Connection Role	A *Role* whose access rights include physically connecting to an OpenEdge database and/or changing an existing connection's user identity
Database connection	A persistent connection of a client to an OpenEdge database server for the purpose of accessing the database stored data. The physical connection to an OpenEdge database uses an OpenEdge security token to grant/deny individual users the ability to establish a connection and/or change its current user identity
Domain	A collection of users, objects, or resources that conform to a common policy
Domain access code	An OpenEdge domain's unique secret key that is used to seal an OpenEdge security token so that it may be authorized by OpenEdge resources
Domain registry	aka OpenEdge Domain Registry, a configuration of OpenEdge domains that includes its authentication system, authentication process options, and a unique domain access code used by OpenEdge resources to authorize access
OpenEdge database enhanced connection security	A collection of interdependent OpenEdge product security features first available in OpenEdge 11.6.2 that provide a greater degree of database connection security
OpenEdge Domain	aka Domain, used to identify a set of users that share a common authentication process and set of rights for access to OpenEdge resources (such as a database)
OpenEdge Authentication Gateway	A Progress Application Server (PAS) for OpenEdge instance configured for secure execution of a *Security Token Service (STS)* application
OpenEdge security token	A *security token* native to all OpenEdge component's *authentication* and *authorization* processes
PAS for OpenEdge	A *Progress Application Server* (*PAS)* extended to execute one or more ABL or WebHandler applications
Progress Application Server (PAS)	A Progress Web application server based on Apache Tomcat that used by multiple Progress products
Protected Resource	A physical something in a computer system whose access is limited to certain authorized users. Examples would be an OS server, database, data records, file system, etc.
RBAC	Acronym for *Role Based Access Control*
Role	A group attribute that binds individual users to a certain set of right(s)
Role Based Access Control (RBAC)	A model for controlling access to protected resources based on a user's granted *role* rather than their individual identity

Role Membership	The inclusion of an individual user in a *role* that will be used by RBAC
Security Token	A *token* that contains verifiable proof of a user's identity and granted role(s), and is used by an authorization process
Security Token Service (STS)	A Web application that runs in a *PAS for OpenEdge* server that provides *authentication* and *security token* services for OpenEdge distributed applications and databases
Spring Security	An industry-recognized authentication and authorization framework used in Java applications, featuring a direct-injection architecture
STS Client	An (authorized) client of an *STS* application
STS Client Key	A key (aka credentials) established by an administrator, holding ownership of a STS Key, for authorizing individual OpenEdge installation's access to an STS application
STS Key	Functionality in an STS application that is used to selectively grant OpenEdge ABL client and/or database rights to use its authentication and security token services
STS Server Key	A key (aka credentials) that uniquely identifies and instance of an STS application, and is used to generate and authorize STS client access via *STS Client Keys*
Server-side security	The *authentication*, *authorization*, and *auditing* processes executed by server processes
Token	Something that uniquely identifies an *authenticated* users identity

2

Configuring the OpenEdge Authentication Gateway

The OpenEdge Authentication Gateway is a Progress Application Server for OpenEdge instance where the OESTS web application (`oests.war`) is deployed. It functions as a Security Token Server (STS) that returns a validated client-principal object in response to client requests for database connections.

After you deploy OpenEdge Authentication Gateway, you can configure the Authentication Gateway by editing a number of JSON files and, optionally, by editing the Spring Security context file.

For details, see the following topics:

- Configuration overview
- Configuring domains
- Configuring event callbacks
- Spring Security Configuration
- Create and deploy a certificate for HTTPS with the OpenEdge Authentication Gateway Server
- STS server key configuration

Configuration overview

OpenEdge Authentication Gateway Server Configuration

Configuring an Authentication Gateway server includes making site-specific changes in three distinct subsystems in the underlying PAS for OpenEdge instance:

1. The server's ports, admin accounts, client connections, and client request execution
2. ABL session management and the Session Manager’s multi-session agent process
3. ABL application event procedures, PROPATH, and any database connections

You can expect to configure these site-specific items:

- HTTPS network port
- HTTPS server private key and digital certificate
- Any optional IP address groups that are allowed to connect
- Any optional automated daemon startup scripts
- Any optional site-specific startup scripts for setting the server’s process environment

To harden the Authentication Gateway, the following changes to the underlying PAS were made:

- Both Tomcat and OpenEdge remote administration web applications are removed. This reduces the Authentication Gateway’s port footprint to just what is needed to service clients and leaves all administration to command line tools. The remote administration web applications may be re-installed at a test-site at an administrator's discretion. (See the PAS for OpenEdge Administration Guide.) However, it is not recommended.
- Standard PAS for OpenEdge administrator user accounts have been hardened by inserting digested passwords into the `conf/tomcat-users.xml` file.
- The ROOT web application and default ABL application have been removed to prevent the injection of other ABL applications. While inserting another ABL application into an Authentication Gateway can be done, it is not recommended.

See the PAS for OpenEdge documentation for more information on all of these sub-systems.

OpenEdge Security Token Service Configuration

The OpenEdge Security Token Service (STS) is the component that performs user-direct login and OS login SSO processes for the OpenEdge database. The STS is deployed into an Authentication Gateway as an ABL application, which means that it has its own configuration for PROPATH, startup parameters, and event handlers. The STS is distinct from any other ABL application that may be deployed into the same server.

You can expect to configure the service’s OpenEdge domains that provide the user-direct login and OS login SSO services, including individual domain settings for:

- The type of external user authentication system to integrate with
- Unique Domain Access Code
- Optional ABL policy module
- Optional ABL policy run-time options
- The type of actions the OpenEdge domain is allowed to execute
- Optional STS Key feature, that provides per OpenEdge installation access to the Authentication Gateway

Configuring domains

A minimal Authentication Gateway configuration requires at least one domain entry. Without domain configuration information, the OESTS implementation will not work. Other domains may be provided, and optionally, ABL policy and event callback handlers may also be defined.

Domain configuration files

Domain configurations are stored in *instance*/webapps/ROOT/WEB-INF/config/domains.json, where *instance* is the name of the instance where your OESTS web application is deployed. This is a simple JSON-formatted text file that can be edited to define domain entries.

Note: Alternatively, you can export domains by dump_domains.p.

Additionally there is a *instance*/webapps/ROOT/WEB-INF/config/domains.keystore file. This is a JKS keystore file that holds the domain access codes for the configured domains. (See Configuring domain access codes on page 27 for more information.)

The domains.json file

The default domains.json file contains blank ("") domain configured for authentication and oests.server, which is disabled by default. You will need to provide any additional domains required in your environment.

The default blank domain is configured to use a property file as the authentication mechanism. This file is stored in *instance*/webapps/ROOT/WEB-INF/config/users.properties. This default domain configuration is simplified intentionally to make initial development easier, but using the blank domain at later stages is not recommended, especially in a multi-domain environment.

A domain configuration defines the authentication provider to be used for login (see Spring Security Configuration on page 34 for more information), the ABL policy for the domain, the event handler implementation, as well as additional context information that can be used by policy and event callback handlers.

The following is a sample domains.json file that defines a single domain given a blank domain name. The domain is enabled and is marked as supporting both Single Sign On (SSO), and login (authenticate) operations. The last required entry is the name of the authentication provider implementation. In this case, it is named properties. The properties authentication provider is configured to use a users.properties file with a couple of simple default user/password combinations for testing purposes.

domains.json

```
{
    "version": "1.0.0",
    "domains": [
     {
     "name" : "",
        "enabled" : true,
        "description" : "Default domain",
        "actions" : {
            "authenticate" : {
             "enabled" : true,
             "options" : ""
            },
            "exchange" : {
             "enabled" : false,
             "options" : ""
            },
            "sso" : {
```

```
             "enabled" : true,
             "options" : ""
            },
            "refresh" : {
             "enabled" : false,
             "options" : ""
            }
       },
       "options" : "",
       "authProvider" : "properties",
       "policyProvider" : "",
       "events" : {
            "provider" : "",
            "groups" : {
                "client" : false,
                "tokenAuthenticate" : false,
                "tokenExchange" : false,
                "tokenRevoke" : false,
                "tokenValidate" : false,
                "tokenRefresh" : false,
                "policy" : false,
                "sessionLogin" : false,
                "sessionLogout" : false,
                "sessionValidate" : false,
                "sessionRefresh" : false
            }
       }
    },
    {
     "name" : "oests.server",
        "enabled" : false,
        "description" : "STS server key domain",
        "actions" : {
            "authenticate" : {
             "enabled" : false,
             "options" : ""
            },
            "exchange" : {
             "enabled" : false,
             "options" : ""
            },
            "sso" : {
             "enabled" : true,
             "options" : ""
            },
            "refresh" : {
             "enabled" : false,
             "options" : ""
            }
        },
        "options" : "",
        "authProvider" : "anonymous",
        "policyProvider" : "",
        "events" : {
            "provider" : "",
            "groups" : {}
        }
    }],
    "policyProviders" : {

    },
    "eventProviders" : {

    }
}
```

Configuration file monitoring

The domains.json, domains.keystore, and users.properties configuration files are actively monitored for changes by the OESTS application. If a change is made to one of these files, the changes will be applied without restarting the OESTS application. The files are read at startup of the OESTS web application and again after the files have changed.

Configuration file location

You can change the location and name of the OESTS configuration files by editing sts.properties file. This file is stored in *instance*/webapps/ROOT/WEB-INF/config/users.properties. By default, configuration files are referenced as relative file paths from the web application. Edits to sts.properties require a restart of the OESTS web application or the Authentication Gateway itself.

Configuring domain access codes

Domain access codes for the OpenEdge Authentication Gateway are stored separately as a Java KeyStore. The Java KeyStore must be built using the OpenEdge gendomreg command.

To create the Java KeyStore, first add the domains and Domain Access Codes (DAC) into a CSV file (for example, domainsks.csv). The CSV file lists OE domains in the *Domain-name*,*Domain-access-code* format. Each domain must be listed on a separate line, and the passwords can be oech1::encoded. For example:

```
mydomain,mypassword
local,localkey
```

Note: Quoted domain names, commented lines (#...), and blank lines are supported by the current version of gendomreg. Older versions do not support these syntactical features.

When the CSV is complete, use gendomreg to create a Java KeyStore file. For example:

```
proenv>gendomreg domainsks.csv domains.keystore
```

After domains.keystore is created, move it to the same directory as the domains.json configuration file (usually *instance*/webapps/ROOT/WEB-INF/config/).

Notes

- The domain names in the domains.keystore and in the domains.json files must match exactly, including the case of letters, or the domain will be disabled for use by the Authentication Gateway.
- There are default domains.keystore and domains.json files in*instance*/webapps/ROOT/WEB-INF/config/. The files are installed with default values suitable for testing purposes.
- The domainsks.csv file is not used by the Authentication Gateway. It is just an intermediate file used by gendomreg to generate the domains.keystore file. You can name this file anything you want. However, because domainsks.csv contains clear text passwords, you should remove it from the OESTS machine and keep a backup of it in a secure location.

Domain configuration requirements for Native Token Exchange (SSO)

Support for native (local desktop) OS logins requires a domain configuration that supports specific options. Specifically, the domain configuration must have:

- A domain type of "_oslocal". This is the authentication provider.
- "exchange" listed as one of the supported actions, and marked as enabled
- "-processid" option listed in the action options for "exchange"

These required settings are shown in bold in the following example.

If any of these are missing from the domain configuration, then token exchange for native OS logins will be rejected. Below is an example domain configuration supporting authentication for OS local logins.

```
{
    "version": "1.0.0",
    "domains": [
        {
        "name" : "local",
        "enabled" : true,
        "description" : "Domain supporting OS local logins",
        "actions" : {
            "authenticate" : {
                "enabled" : true,
                "options" : ""
            },
            "exchange" : {
                "enabled" : true,
                "options" : "-processid"
            },
            "sso" : {
                "enabled" : false,
                "options" : ""
            },
            "refresh" : {
                "enabled" : false,
                "options" : ""
            }
       },
       "options" : "",
       "authProvider" : "_oslocal",
       "policyProvider" : "",
       "events" : {
            "provider" : "",
            "groups" : {
                "client" : false,
                "tokenAuthenticate" : false,
                "tokenExchange" : false,
                "tokenRevoke" : false,
                "tokenValidate" : false,
                "tokenRefresh" : false,
                "policy" : false,
                "sessionLogin" : false,
                "sessionLogout" : false,
                "sessionValidate" : false,
                "sessionRefresh" : false
            }
       }
    }],
    "policyProviders" : {
             },
    "eventProviders" : {
    }
}
```

Configuring LDAP Connections in a Security Token Service (STS)

Configuring an LDAP (Active Directory) service as an OpenEdge domain's point of user account authentication requires obtaining information about the directory service and then translating that information into a set of STS configuration properties.

It helps if you have some familiarity with directory services, particularly regarding with their organization, the stored object's attribute names, and how to search for user accounts and other objects. The following provides some of that information and may shorten the time needed to successfully authenticate users to an LDAP service.

There is always a core set of information that you will need to obtain to configure and test user authentication via an LDAP service. Because each LDAP directory service's hierarchical tree of objects (including user account objects) is customized for each deployment site, you must obtain information from the LDAP service administrator. All of the core information will be applied to properties (`sts.ldap.`*xxxx*) used by the STS's LDAP authentication process. For example:

- The LDAP directory service's URL, including the hostname, port and base-dn information:

```
sts.ldap.context.providerUrl=ldap://host[:port][/base-dn]
```

- The DN (Distinguished Name) of an existing LDAP user account that may be used to search for an locate the user account being authenticated:

```
sts.ldap.context.userDn=ldap-user-dn
sts.ldap.context.password=ldap-user-password
sts.ldap.context.userDn=ldap-user-dn
sts.ldap.context.password=ldap-user-password
```

- The information needed to build the LDAP search query used to locate the LDAP account of the user being authenticated. The search query requires the location (DN) of where to begin the search and the LDAP user account attribute name that contains the user-id of the user being authenticated:

```
sts.ldap.user.searchBase=search-base-dn
sts.ldap.user.searchFilter=(attr-name={0})
```

Note: Common *attr-name* values may be *uid* for UNIX LDAP servers or *sAMAccountName* for Windows Active Directory. Your LDAP administrator can access the attribute names for user accounts.

- The STS LDAP user authentication requires that the LDAP user account (DN) be a member of at least one LDAP group object. So you must supply the staring location of where to begin an LDAP search, the group object's attribute name holding the authenticated user's DN, and the group object's DN field name holding the group (Role) name:

```
sts.ldap.group.searchBase=search-base-dn
sts.ldap.group.searchFilter=(group-attr-name={0})
sts.ldap.group.roleAttribute=cn
```

An example configuration may look like:

```
sts.ldap.context.providerUrl=ldap://ldap.acme.com:389
sts.ldap.context.userDn=uid=admin,ou=system
sts.ldap.context.password=secret
sts.ldap.user.searchBase=ou=users,o=acme,dc=acme,dc=com
sts.ldap.user.searchFilter=(uid={0})
sts.ldap.user.searchSubtree=true
sts.ldap.group.searchBase=ou=groups,o=acme,dc=acme,dc=com
sts.ldap.group.searchFilter=(uniqueMember={0})
sts.ldap.group.roleAttribute=cn
```

Debugging an STS LDAP authentication involves enabling logging in the STS's logging configuration file *logging.xml* and adding the entry:

```
<logger name="org.springframework.security.ldap" level="TRACE" />
```

Configuring policies

A domain policy is an ABL-based implementation that enforces sign on and authentication policies. A domain policy configuration is stored in the `domains.json` configuration file.

Each domain configuration can reference a single domain policy configuration. The policy configuration requires the fully qualified name of an ABL class that implements `OpenEdge.Security.STS.IPolicyProvider`. (Optionally, an MD5 file, taken from `r-codeinfo:md5-value`, can be included to enforce checksum requirements for the policy implementation class.)

The policy class (or rcode) needs to be found in the PROPATH. The recommended location is under `instance/webapps/ROOT/WEB-INF/openedge`, which is in the PROPATH by default.

For example:

```
    "version": "1.0.0",
    "domains": [
        {
        "name" : "local",
        "enabled" : true,
        "description" : "O/S Authentication",
        "actions" : {
            "authenticate" : {
                "enabled" : true,
                "options" : ""
. . .
        },
        "options" : "-processid",
        "authProvider" : "_oslocal",
        "policyProvider" : "login",
        "events" : {
            "provider" : "",
            "groups" : {}
        }
    },
. . .

"policyProviders" : {
    "login" : {
        "type" : "com.progress.sts.SampleLoginPolicy",
        "hash" : ""
    }
},
```

You need to then define the policy name under the domain configuration. For the example above, it is the bold line, `"policyProvider" : "login"`.

The following is a sample login policy class. It prints some messages to the log and adds a property to the client-principal object that is returned to the client.

SampleLoginPolicy.cls

```
using Progress.Lang.*.
using OpenEdge.Security.STS.IPolicyProvider.
using OpenEdge.Security.Principal.
using Progress.Json.ObjectModel.JsonObject.
using OpenEdge.Security.PAMStatusEnum.block-level on error undo, throw.
class com.progress.sts.SampleLoginPolicy implements IPolicyProvider:
    method public PAMStatusEnum ApplyPolicy( input pcSender as character,
                                             input pcPolicy as character,
                                             input phClientPrincipal as Principal,
                                             input pcDomainCtx as JsonObject,
                                             output pcStatusDetail as character ):
        message "sender:" pcSender skip
                "policy:" pcPolicy skip
                "C-P Token" phClientPrincipal:Token skip
                "context:" pcDomainCtx.
        pcStatusDetail = "OK".
        return PAMStatusEnum:Success.
    end method.
end class.
```

Configuring event callbacks

Event callback policies are stored in the `domains.json` configuration.

The following is a sample event configuration file containing a single event callback configuration named `login`, which is implemented by the `SampleEventHandler` class. Event handler classes must implement `OpenEdge.Security.STS.IEventProvider`.

```
    "version": "1.0.0",
    "domains": [
        {
        "name" : "local",
        "enabled" : true,
        "description" : "O/S Authentication",
        "actions" : {
            "authenticate" : {
                "enabled" : true,
                "options" : ""
. . .
        },
        "options" : "-processid",
        "authProvider" : "_oslocal",
        "policyProvider" : "",

        "events" : {
            "provider" : "login",
            "groups" : {
               "tokenAuthenticate" : true,
                "tokenExchange" : true
}
        }
    },

. . .

"eventProviders" : {
    "local" : {
        "type" : "com.progress.sts.SampleEventHandler",
        "hash" : ""
    }
},
```

In the above code example, the bold lines:

- **`"events": { "provider" : "login"`** — references the eventProvider to be called when the group action happens in the domain
- **`"tokenAuthenticate" : true, "tokenExchange" : true`** — the events that will cause the eventProvider to run the appropriate code
- **`"local" : { "type" : "com.progress.sts.SampleEventHandler", "hash" : ""`** — this is the event run for the “local” domain when tokenAuthenticate/Exchange happens

The following is a sample event handler class that prints messages to the log file.

SampleEventHandler.cls

```
using Progress.Lang.*.
using OpenEdge.Security.STS.IEventProvider.
using OpenEdge.Security.Principal.
using Progress.Json.ObjectModel.JsonObject.
block-level on error undo, throw.
class com.progress.sts.SampleEventHandler implements IEventProvider:
    method public void RecordEvent( input pcSender as character,
                                    input pcEvent as character,
                                    input poPrincipal as Principal,
                                    input poDomainCtx as JsonObject ):
    message "sender:" pcSender skip
    "event:" pcEvent skip
    "C-P Token" poPrincipal:Token skip
    "context:" poDomainCtx.
    end method.
end class.
```

Hash values for events and policies

The “hash” value in the code examples is a security feature to make sure the correct code is being run. If the hash doesn’t match, it may mean the code was replaced, and the code will not run. This protects from someone changing the policy code (and possibly changing the client-principal) or dumping private information in the log file during an event.

Follow these steps to use the “hash” value protection:

1. Compile the code with MD5 hash, for example:

```
COMPILE SampleLoginPolicy.cls GENERATE-MD5 SAVE.
```

2. Get the MD5 hash value from the compiled code, for example:

```
RCODE-INFO:FILE-NAME = "SampleLoginPolicy.r".
DISPLAY RCODE-INFO:MD5-VALUE format "x(44)".
```

3. Deploy the r-code only to the Authentication Server’s PROPATH
4. Add the hash value to the `domains.json` file

Spring Security Configuration

The Authentication Gateway Spring Security implementation is configured in *gateway_instance*`/webapps/ROOT/WEB-INF/sts-context.xml`. It is a standard Spring Security XML configuration file that contains STS configuration details. (Note that the Authentication Gateway is an OpenEdge implementation of an STS.) Usually, you will only need to modify the STS configuration in this file if you choose to add a different authentication provider.

The following shows the section of `sts-context.xml` that specifies the default authentication providers:

sts-context.xml

```
<b:property name="authenticationProviders">
    <b:map>
        <b:entry key="ldap" value-ref="ldapAuthProvider"></b:entry>
        <b:entry key="_oslocal" value-ref="OSLocalAuthProvider"></b:entry>
        <y>
    </b:map>
</b:property>
```

This section shows the default implementations for LDAP and oslocal logins. The domains configured in the `domains.json` file reference these authentication providers by name. Each provider is expected to implement the standard Spring `IAuthenticationProvider` interface.

The following is a brief description of each of the default authentication providers:

- `ldapAuthProvider` — Makes use of the standard spring LDAP authentication provider implementation and requires an LDAP server configuration.
- `OSLocalAuthProvider` — Performs authentication of login requests (a client-principal containing a user/password combination) for authentication by the local operating system.

For more information about Spring Security configuration, see the Spring Security documentation online.

Create and deploy a certificate for HTTPS with the OpenEdge Authentication Gateway Server

The OpenEdge Authentication Gateway server is accessed by encrypted HTTPS calls.

Use the following steps to create and deploy a certificate for HTTPS with OpenEdge Authentication Gateway Server:

1. Create a valid certificate using the `pkiutil` utility. See *OpenEdge Getting Started: Installation and Configuration* for details and complete syntax.

   The result of running `pkiutil` is:

   - Certificate Authority certificate (`CA.cer`)
   - Public certificate (`name.cer`)
   - PEM file (`name.pem`)

2. Import the certificates for use by OpenEdge executables using `certutil` that place hashfiles in the appropriate `%DLC%` directory, as shown:

   - Run `certutil import CA.cer`

     This creates a `%DLC%/certs/`*`hashfile`*`.0`
   - Run `certutil import name.cer`

     This creates a `%DLC%/certs/`*`hashfile`*`.0`
   - Copy `name.pem` to `%DLC%/keys`

   Repeat this step on all database installations accessing the OpenEdge Authentication Gateway server.

3. Change the keystore for the OpenEdge Authentication Gateway server to use the new certificate, as follows:

   a. Save the current keystore to a backup. For example:

   ```
   mv tomcat-keystore.p12 tomcat-keystore.p12.orig
   ```

   b. Create a new keystore. For example:

   ```
   sslc pkcs12 -export -in %DLC%/keys/name-systest.pem -out tomcat-keystore.p12 -name
    ux-systest
   ```

   You will be prompted for:

   - Pass phrase for the `.pem` file
   - Export password
   - Verification of export password

   c. Verify the keystore. For example:

   ```
   keytool -list -v -keystore tomcat-keystore.p12 -storetype pkcs12
   ```

4. Modify the OpenEdge Authentication Gateway Server to load the new keystore using the `tcman config` utility. For example:

   ```
   tcman config psc.as.https.keypass=password
   tcman config psc.as.https.keyalias=name
   ```

5. Restart the OpenEdge Authentication Gateway Server.

STS server key configuration

An optional STS server key file can be installed to validate client requests to ensure that they are coming from authorized database client installations. By default, the Authentication Gateway does not require the STS server key file to perform authentication and SSO token exchange.

The STS server key file is generated by the `stskeyutil` command line utility installed with OpenEdge. For more information on `stskeyutil`, see STS Key Utility (stskeyutil) on page 78.

The following sections describe how to configure the Authentication Gateway with an STS server key, and how to install STS client keys.

Generating an STS server key

The first step is to generate a new STS server key using the `stskeyutil` command. For example:

```
>stskeyutil create
>Enter server key file password:
>Enter server key's domain access-code:
>created server key file: oests-key.ecp
```

Configuring an STS server domain access code

You must create a STS domain configuration that will be used by the STS server to store and access the domain access code for the server key in addition to any domains used for client authentication. (See the `domain.json` example in Configuring domains on page 25.)

Note the following:

- The name of the domain should be `oests.server`. (The "enabled" property in the domain configuration should be changed from `false` to `true`.) If you choose a different domain name, be sure to update the `sts.properties` file. Set the new domain by editing the `sts.server.keydomain` property.

  You must also specify the domain name via `stskeyutil create`.

- The domain must have the SSO domain action.
- The domain must specify the `anonymous` domain type.
- When configuring the domain, provide the domain access code in `domainsks.csv` . The domain access code must exactly match the domain access code used when generating the STS server key. Run `gendomreg domainsks.csv domains.keystore` and then copy `domains.keystore` to the *`instance`*`/webapps/ROOT/WEB-INF/config` directory.

Configuring the Authentication Gateway with the STS server key

Once the key is generated, and the domain configuration added, you must enable the STS server key on the Authentication Gateway . The Authentication Gateway configuration must be modified both to turn on STS key support explicitly, and to reference the server key file location.

At a minimum, you must modify the following required properties in the `sts.properties` file:

- Set `sts.server.key.enabled` to true.
- Set `sts.server.keyfile` to the location of the server key file. This may be absolute or relative directory (i. e. relative to the PAS for OpenEdge working directory).

  Note: Due to how Spring configuration interprets property files, on Windows you must use forward slashes (`/`)for any file path separators for the key file name, or use double-back slashes (`\\`)to escape them.

- Set `sts.server.keyfilepassword` to the password used to generate the server key file. Progress Software Corporation strongly recommends that the password be `oech1::` encoded.

For example:

sts.properties snippet

```
sts.server.key.enabled=true
sts.server.keyfile=oests-key.ecpsts.server.keyfile
password=oech1::20333c34252a2137
```

Client key configuration

Once the STS server is configured you must configure the client with an STS client key generated from the STS server key.

To install the client key, run `stskeyutil install`. You must run this utility on the client machine with the OpenEdge install on that machien that will try to connect to the STS server. For example:

```
>stskeyutil install -url https://acme.com:8443/oests -overwrite
>Enter server key file password:
created client key file
```

Note:

The `-url` switch is mandatory and must reference the appropriate Authentication Gateway.

The client key file will be installed under `%DLC%/keys` by default.

The URL is hashed to generate the STS client key file name in the keystore. Each Authentication Gateway has a unique URL that is referenced by the database domain configuration. The STS client key installation must match this value exactly.

Start the Authentication Gateway

Once the STS server key has been installed and configured you can start the Authentication Gateway (or PAS for OpenEdge instance) where STS web application is installed.

For example:

```
oeauthserver/bin/tcman.sh start
```

Once the server starts you should see a message similar to the following in the server log file. You will need to adjust the logging level of the OESTS web application to see it. Edit `ROOT/WEB-INF/logging.xml` to increase logging level.

```
STS server key is enabled. Server key file configuration is required.
Clients will be required to be configured with the matching client key.
```

Testing the client key

Run the `stsclientutil` utility to test the client key installation. For example:

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

For more information on `stsclientutil`, see STS Client Utility (stsclientutil) on page 75.

Troubleshooting

Check the STS server log file in the Authentication Gateway instance's `/logs/*.log` for errors.

Normally STS web application logging is set to `WARNING`.You can get ping status messages in the STS log file if you increase the logging level. Edit `ROOT/WEB-INF/logging.xml` to increase the logging level.

The ping status messages in the log file will look similar to the following:

```
Ping succeeded. Received request with a valid client key header.
Ping succeeded. Received client request with a valid client key header, but client key
 header is not required.
Ping failed. Received request without the required client key header.
Ping failed. Received client request with an invalid client key header.
Ping succeeded. Received request without a client key header. Request header is not
required.
```

See *Progress Application Server for OpenEdge: Administration Guide* for more information about log files and logging levels.

You can also use the `-l 5` switch with `stsclientutil` to make the logging messages more verbose. However, `ping` returns very little information. It is primarily used to test if the installed client key, for a given URL, matches the server key that is installed with the Authentication Gateway.

Other STS key properties

The following table lists other properties in the `sts.properties` file that are related to STS keys:

Property	Default	Description
`sts.server.ststokenname`	`x-oests-token`	Identifies the HTTP header that the Authentication Gateway looks for to identify the client. Can be changed to any string, but the client configuration must match. This value must match the name of the header name in the STS server key, if one is set up via `stskeyutil`.
`sts.server.previouskeyfile`	`<blank>`	(Optional) Specify the file name of the secondary STS server key. Used when rolling out STS client key updates incrementally. If left blank, there is no secondary STS server key.
`sts.server. previouskeyfilepassword`	`<blank>`	Secondary server key password.
`sts.server.previouskeydomain`	`oests.server`	A domain name that indicates which domain name to use to perform client key validation. Must match the value stored in the previous STS server key file.
`sts.libstskey.loglevel`	`2`	Logging level for `libstskey.dll`.
`sts.libstskey.logfile`	`${catalina.base}/logs/libstskey.log`	Log file for `libstskey.dll` output.

3

Configuring your OpenEdge Database to use the OpenEdge Authentication Gateway

The securing your OpenEdge database with the OpenEdge Authentication Gateway controls access to data with strong user authentication and connection authorization processes.

The following sections describe the steps to configure an OpenEdge database to communicate with the STS.

For details, see the following topics:

- Configuring your OpenEdge database
- SSO Token Exchange
- STS Client Key (for database server)
- Start OpenEdge database server

Configuring your OpenEdge database

This section describes the steps to configure an OpenEdge database to communicate with the STS.

Domain configuration

Prior to enabling your database to use the Authentication Gateway, you must have domains defined and loaded in the database that match the domains defined in the STS. If all your domains are currently defined in your database, you can dump them from your database with `dump_domains.p` to import into your STS; conversely you can load domain definitions into your database with the dictionary `prodict/load_d.p`.



Warning: You must add domains to the database before you enable the database to use the Authentication Gateway, so that you are not locked out.

Once you have your domains in agreement between the Authentication Gateway STS server and the database, if you changed domain information in the STS, re-generate an STS Server key. See STS server key configuration on page 36.

Load URL of Authentication Gateway STS server

The database must know the URL of the Authentication Gateway STS server. Load the URL of the STS into the database with the STS URL Utility, as shown:

```
stsurlutil update -url url
-db dbname [db-options]
```

Specify the new URL for the Authentication Gateway to insert into the database configuration using the format: `https://<host>[port][/sts-application-name]`

For more details on the `stsurlutil`, see STS URL Utility on page 91.

Connection roles

If you are using roles, you must enable the database and then grant users roles using the STS Connection Role utility.

To enable connection roles, use the following command:

```
stsconnroleutil enable -db dbname
```

To grant a user connection roles, use the following command:

```
stsconnroleutil grantuser -user username
        -can { yes | no } -db dbname
```

For details on `stsconnroleutil`, see: STS Connection Role Utility (stsconnroleutil) on page 83

Enable the database to use the Authentication Gateway

Once you have added domains and the STS URL to your database, you can enable it to use the Authentication Gateway. Use the following command:

```
proutil dbname -C enableauthgateway
```

For more details on the command, see PROUTIL ENABLEAUTHGATEWAY qualifier on page 71

You can disable the use of the Authentication Gateway at any time with PROUTIL DISABLEAUTHGATEWAY. However, once you disable use of the Authentication Gateway in your database, you must re-configure the STS URL prior to running PROUTIL ENABLEAUTHGATEWAY. The PROUTIL DISABLEAUTHGATEWAY deletes the URL from the database. For more information, see PROUTIL DISABLEAUTHGATEWAY qualifier on page 71.

SSO Token Exchange

This section describes connecting to your OpenEdge database using your operating system login id.

First, in the STS, you need the domain to contain the EXCHANGE action, and the "-processid" domain option, similar to how the 'local' domain is set up. More details can be found at Configuring domains on page 25. For example:

```
{
"name" : "local",
"enabled" : true,
"description" : "O/S Authentication",
"actions" : {
    "authenticate" : {
        "enabled" : true,
        "options" : ""
    },
    "exchange" : {
        "enabled" : true,
        "options" : "-processid"
    },
    "sso" : {
        "enabled" : true,
        "options" : ""
    },
    "refresh" : {
        "enabled" : false,
        "options" : ""
    }
},
"options" : "",
"authProvider" : "_oslocal",
"policyProvider" : "",
"events" : {
    "provider" : "",
    "groups" : {}
}
```

Note the boleded line that adds `-processid` to the options section in the local domain.

When connecting to the database, you can use one of the following:

- -OSUser -domain local
- -OSUser -U *<OS-user-name>*@local,where *<OS-user-name>* must match the operating system user name currently running the ABL client

You can use any domain that is set up with the _oslocal provider for SSO token exchange (not just 'local') as long as the domain is configured as mentioned above in the `domains.json` file. That includes the "" domain, in which case you connect like this:

- -OSUser
- -OSUser -U *<OS-user-name>*
- -OSUser -U *<OS-user-name>*@
- No additional parameters is implied as -OSUser

STS Client Key (for database server)

In order for your OpenEdge database to use the Authentication Gateway, it must have an STS client key that corresponds to an STS Server Key from the STS server. If you have not previously created an STS Server key, follow the steps at STS server key configuration on page 36.

If your database and STS server are not running on the same machine, copy the `oests-key.ecp` file from the STS server machine to the database server machine.

Install an STS client key

After you have an STS server key, create a client key with `stskeyutil` as shown:

```
stskeyutil install -url STS-URL -file keystore-dir/oests-key.ecp
```

The value of *STS-URL* must match exactly what is stored in the database from the `stsurlutil` command). You are prompted for the same password entered when you created the STS Server key.

Specify the location of the STS server key with `-file` *keystore-dir*.

By default, the hashed key is placed in `%DLC%/keys`.

Note: You can specify a different path with `-keystorepath`. You CANNOT move the client key file to a different machine or different OpenEdge installation, even if it's on the same machine.

Testing the key

You can run `stsclientutil ping` to validate that the key is validated on the STS side, using the exact same URL, as shown:

```
stsclientutil -cmd ping -url STS-URL
```

Start OpenEdge database server

Once you have created an STS client key and configured and enabled the database to use the Authentication Gateway, you can start the server (or connect single-user).

Start the database, as shown:

```
proserve dbname S 12345
```

If the STS client key is in the default location, you need no additional parameters.

If you are going to be connecting Client-Server, you need to generate the client key on both client and server on both.

The `oests.<date>.log` file contains messages about the STS client key validation and if anything goes wrong with the validation. You can also increase (or decrease) logging messages in the database log file with the startup parameter `-stsclientlogging`. See STS logging level (-stsloggінglevel) on page 70.

Additional database startup parameters to keep in mind include:

- `-Mm` cannot be set to the minimum (350) when using the Authentication Gateway STS server. You must set it to at least 360. The default is 1024 and the maximum is 32,600
- `-secsize` specifies the sized of the security cache that can help connection performance. See Security cache size (-secsize) on page 69
- `-certstorepath`, `-keystorepath`, `-nohostverify`, and `-nosessionresue` are shared between SSL-enabled connections and communication with the Authentication Gateway STS server

You can perform authentication or exchange using `stsclientutil` if you want to perform a simple test and take the database out of the picture (change the parameters for credentials as needed, `-U`, `-P`, `-D`), as shown:

```
stsclientutil -cmd authenticate -url STS-URL -user test@seal -password test
  stsclientutil -cmd exchange -url STS-URL -domain local
```

For more information, see STS Client Utility (stsclientutil) on page 75.

4

Reference entries

This chapter contains reference entries for:

- ABL reference
- Startup parameters
- Database utilities
- OpenEdge Authentication Gateway utilities

For details, see the following topics:

- ABL reference
- Startup parameters
- Database utilities
- OpenEdge Authentication Gateway utilities

ABL reference

The sections that follow contain new and updated ABL constructs.

SET-DB-CLIENT function (enhanced for STS)

What's new

For an STS-enabled database, SET-DB-CLIENT does not check the user identity against the ABL session domain registry. In this case, the identity is always checked against the database's own domain registry configuration.

Description

Sets the user identity for a specified connected OpenEdge database using an unsealed or a sealed client-principal object.

For an unsealed client-principal object (in the INITIAL state), this function performs a user authentication operation on the user identity asserted by the object. If successful, the function seals the client-principal and assigns the user identity to a specified database connection, and if the database is multi-tenant this also sets the user's tenancy.

For a sealed client-principal object (in the LOGIN state), this function performs a single sign-on (SSO) operation to validate the user identity represented by the object. If successful, the function assigns the user identity to a database connection, and if the database is multi-tenant this also sets the user's tenancy.

If the user identity is set successfully and the database is multi-tenant, the connection accesses the database through the tenant organization that is configured for the user's domain, and the tenant information is added to the client-principal object used to set the identity.

The function returns TRUE if the user identity is set successfully; if unsuccessful, the function returns FALSE, and the connection identity remains unchanged.

Note: Using this function overrides any user identity previously set for the database connection by either the SECURITY-POLICY:SET-CLIENT( ) method or the SETUSERID function. (See *OpenEdge Development: ABL Reference* for details.)

Syntax

```
SET-DB-CLIENT ( { client-principal-handle | ? } [ , { integer-expression |
logical-name | alias | ? } ] )
```

client-principal-handle

A handle to a client-principal object. If the client-principal object is unsealed, it must be initialized with the attribute values required by the SEAL( ) method (enhanced for STS) on page 58 in addition to any PRIMARY-PASSPHRASE attribute value required to authenticate the asserted user identity. If the object is sealed, it must be sealed with a domain access code that is the same as the access code configured for the user's domain stored in the domain registry trusted by the specified database connection. If this parameter is set to the Unknown value (?), the current identity of affected database connections remains unchanged, and the function unlocks and allows the connection identity to be set using the SET-CLIENT( ) method, as described for the setting of database parameters.

integer-expression | *logical-name* | *alias* | ?

Specifies a database connection on which to set the identity as follows:

integer-expression

The sequence number of a connected OpenEdge database for which to set the user identity. For example, `SET-DB-CLIENT(hCP, 1)` sets the user identity associated with the specified client-principal object for the first database connection, `SET-DB-CLIENT(hCP, 2)` sets the user identity for the second database connection, and so on. If you specify a sequence number that does not correspond to a connected database, the function returns FALSE.

logical-name | *alias*

The logical name or alias of a connected OpenEdge database for which to set the user identity. These forms require a quoted character string or a character expression. If you specify a logical name or alias that does not correspond to a connected database, the function returns FALSE.

If you pass a valid option to identify an OpenEdge database connection, the function operates on that OpenEdge database connection as follows:

- If the LOGIN-STATE attribute on the client-principal object is set to `"INITIAL"`, the function performs a user authentication operation, and if successful, sets user identity for the database connection,
- If the LOGIN-STATE attribute on the client-principal object is set to `"LOGIN"`, the function performs a validation (SSO) of the user identity, and if successful, sets the user identity for the database connection.
- If *client-principal-handle* is the Unknown value (`?`), the current connection identity remains unchanged, and the function removes the SECURITY-POLICY:SET-CLIENT( ) method lockout on the database connection.

If you do not pass an option to identify the specific database connection or you specify the Unknown value (`?`), the function operates on all OpenEdge database connections as follows:

- If the LOGIN-STATE attribute on the client-principal object is set to `"INITIAL"`, the function performs a user authentication operation for the first OpenEdge database connection, and if the authentication is successful, the function:
  1. Sets that database connection's user identity.
  2. With the client-principal LOGIN-STATE attribute set to `"LOGIN"`, performs an SSO operation to validate and (if successful) set the user identity for each remaining OpenEdge database connection in the ABL session.
- If the LOGIN-STATE attribute on the client-principal object is set to `"LOGIN"`, the function performs an SSO operation to validate and (if successful) set the user identity for each OpenEdge database connection in the ABL session.
- If *client-principal-handle* is the Unknown value (`?`), the current identity remains unchanged for all OpenEdge databases connections, and the function removes the SECURITY-POLICY:SET-CLIENT( ) method lockout on all the connections.

Example

To use the `_login.p` procedure that is provided with ABL, you must define user IDs and passwords for users who are authorized to access the database in user accounts associated with authentication-enabled domains. The following sample procedure (`r-login1.p`) is simplified from the `_login.p` procedure provided with ABL.

r-login1.p

```
/* Prompt user for userid and password and set the userid */
DEFINE VARIABLE id       AS CHARACTER FORMAT "x(255)" VIEW-AS FILL-IN
                            SIZE 50 BY 1 LABEL "User id" NO-UNDO.
DEFINE VARIABLE password AS CHARACTER FORMAT "x(255)" VIEW-AS FILL-IN
                            SIZE 50 BY 1 LABEL "Password" NO-UNDO.

DEFINE VARIABLE tries    AS INTEGER         NO-UNDO.
DEFINE VARIABLE lFound   AS LOGICAL         NO-UNDO.
DEFINE VARIABLE hCP      AS HANDLE          NO-UNDO.

/* Note: No test for _User records, as in earlier releases:
   As of Release 11.0, OpenEdge can authenticate to other
   user account systems in addition to the _User table.   */

CREATE CLIENT-PRINCIPAL hCP. /* create a CLIENT-PRINCIPAL only once during login*/

DO ON ENDKEY UNDO, RETURN:  /* return if they hit endkey */
  /* Reset id and password to blank in case of retry */
  ASSIGN
    id       = ""
    password = "".
  UPDATE SPACE(2) id SKIP  password BLANK
    WITH CENTERED ROW 8 SIDE-LABELS ATTR-SPACE
    TITLE " Database " + LDBNAME("DICTDB") + " ".

  /* Use SET-DB-CLIENT instead of SETUSERID */
  hCP:INITIALIZE(id,?,?,password).

  IF NOT SET-DB-CLIENT(hCP,"DICTDB") THEN DO:
    MESSAGE "Sorry, userid/password is incorrect.".
    IF tries > 1 THEN QUIT. /* Only allow 3 tries */
    tries = tries + 1.
    UNDO, RETRY.
  END.

END.

DELETE OBJECT hCP.
hCP = ?.

HIDE ALL.
```

This procedure uses the SET-DB-CLIENT function to authenticate the prompted user identity passed in through the client-principal (`hCP`) to set the connection identity for the database with the logical name, `dictdb`.

If the value of the function is FALSE, the procedure allows the user another try. The user has three tries to log in. The first time, the `tries` variable is 0; `tries` is 1 the second time, and 2 the third. The third time, `tries` is greater than 1 and the procedure exits without any further attempts. If function value is TRUE, the database connection is set to the prompted user identity. In either case, the procedure exits after deleting the client-principal object and hiding the prompt.

Notes

- For any errors while operating on database connections, SET-DB-CLIENT returns FALSE, records any errors in the ERROR-STATUS system handle, and leaves the current identity for a given database connection unchanged. Also, a validation failure when setting the identity for a database connection does not stop SET-DB-CLIENT from operating on other connected databases. It records an error message for that database and attempts the operation on the next one.
- Within a transaction on a:
  - **Multi-tenant database** — Any attempt to set an identity for the connection that changes the current database tenancy raises a run-time error.

  - **Non-multi-tenant database** — As a best practice, Progress Software recommends that you **not** set a new identity for the connection.

- To assign a user identity while making a database connection, use the CONNECT statement.
- This function also checks the value of the LOGIN-EXPIRATION-TIMESTAMP attribute on the client-principal object. If the timestamp has expired before the function can seal (during user authentication) or validate (during SSO) the object, the function sets the LOGIN-STATE attribute to `"EXPIRED"` and returns FALSE without changing the current connection identity.
- To set a connection identity through a user authentication operation:
  - The client-principal object must be **unsealed** (in the INITIAL state).
  - The required attributes must be properly set. See the SEAL( ) method (enhanced for STS) on page 58 and PRIMARY-PASSPHRASE attribute (see *OpenEdge Development: ABL Reference*).
  - The domain in the registry trusted by the database connection must be authentication-enabled. This can include a domain configured with a user-defined authentication system that has an ABL authentication plugin enabled. For information on OpenEdge support for domains and domain configuration, see *OpenEdge Getting Started: Identity Management*.

  When the user authentication operation fails for a database connection, for auditing purposes, this function implicitly executes the AUTHENTICATION-FAILED( ) method on the client-principal and leaves the previous connection identity unchanged.
- To set a connection identity through an SSO operation:
  - The client-principal object must be **sealed** and set to the LOGIN state either explicitly by the application (using the SEAL( ) method) or implicitly by OpenEdge.
  - The object must be valid according to the domain registry used by the database connection: it must be sealed using the access code defined for a registry domain whose name matches the domain name stored in the object.
  - The domain in the registry trusted by the database connection must be enabled and configured with an authentication system that supports SSO.
- To set a connection identity when the authentication system is your ABL application, you must manually authenticate the user credentials for the client-principal object, explicitly call the SEAL( ) method to seal the object, and perform an SSO operation using this function to set the connection identity.
- For an STS-enabled database, this function always checks the user identity against the database's own domain registry configuration.
- For databases not using STS, the domain registry this function uses to perform a user authentication or SSO operation depends on the database's own domain registry configuration. If it is configured to use the application domain registry, the function checks the user identity against the ABL session domain registry loaded using the SECURITY-POLICY:LOAD-DOMAINS( ) method. If it is configured **not** to use the application domain registry, the function checks the user identity against the database's own local domain registry that OpenEdge loads for the connection. For more information on domain and domain registry configuration, see *OpenEdge Getting Started: Identity Management*.
- After a user identity is set for a database connection, the AVM uses that identity to determine if the user has permission to access tables and fields in that particular database.
- If you do not specify a database or pass the Unknown value (`?`), the user identity that this function sets for all connected databases overrides the user identity previously set by a call to the SET-CLIENT( ) method.
- Any database connection whose user identity is set using this function locks out the SET-CLIENT( ) method from setting an identity for that connection until you use this function to unlock it.

- You can also use this function, instead of the SETUSERID function, to set the user identity for a database connection whether or not the user account is in the `_User` table.
- Calling this method generates an audit event, and creates an audit record for the event in all connected audit-enabled databases according to each database's current audit policy settings.
- This function returns FALSE and with messages recorded in the ERROR-STATUS system handle when:
  - A database connection is specified, the client-principal object is sealed, and SSO validation of its identity for the specified database connection fails.
  - No database connection is specified, the client-principal object is sealed, and SSO validation of its identity for any connected database fails. In this case, ERROR-STATUS records a message for each database connection that fails SSO validation.
  - The client-principal object is unsealed and the domain configuration is restricted to SSO operations.
  - A database connection is specified, the client-principal object is unsealed, and user authentication of its asserted identity on the specified database connection fails.
  - The client-principal object is sealed and the domain configuration is restricted to user authentication operations.
- This function does not attempt set the connection identity for the foreign data source of a DataServer connection. However, it does attempt to set the connection identity for the OpenEdge schema holder database.

See also

Client-principal object handle, SEAL( ) method, SET-CLIENT( ) method, SETUSERID function in *OpenEdge Development: ABL Reference*

SETUSERID function (enhanced for STS)

What's new

When using an STS-enabled database, the `_oeusertable` authentication system is not required, and accounts do not have to be in the `_User` table.

Description

Authenticates a user identity for a specified database connection, verifying that the user ID and password supplied to the SETUSERID function match a user account in the `_User` table of the database. If they match, the database connection is set to the specified user identity and the function returns a TRUE value. If the database is multi-tenant, is also sets the user's tenancy.

If the user ID is not in the `_User` table or the password is incorrect, SETUSERID returns a FALSE value and does not assign the user identity to the database connection. You can also check the ERROR-STATUS system handle for any messages returned.

Note:

Using this function overrides user identity previously set for the database connection by either the SECURITY-POLICY:SET-CLIENT( ) method or the SET-DB-CLIENT function.

This function authenticates user identities only against user accounts defined in the `_User` table of a connected OpenEdge RDBMS. To authenticate against additional OpenEdge-supported and application-defined authentication systems, use the SET-CLIENT( ) method or the SET-DB-CLIENT function instead.

Syntax

```
SETUSERID ( userid , password [ , logical-dbname] )
```

userid

A literal value, field name, variable name, or expression that results in a character value that represents the user's user ID. If you use a literal value, you must enclose it in quotation marks ("").

If the user is defined in a non-default domain, this value must be a fully qualified user ID, including both the non-qualified user ID and domain separated by a domain delimiter (@). If the user is defined in the default (blank) domain, only a non-qualified user ID (without a domain delimiter) must be specified. For more information on specifying a fully qualified user ID, see the reference entry for the QUALIFIED-USER-ID attribute.

Note: The default blank domain provides backward compatibility with OpenEdge releases that do not support domains as part of a user's identity.

password

A literal value, field name, variable name, or other character expression that results in a character value that represents the user's password. If you use a literal value, you must enclose it in quotation marks ("").

Alternatively, you can encrypt the password using the ENCRYPT-AUDIT-MAC-KEY( ) method. For more information, see the documentation on encrypted passwords in *OpenEdge Development: Programming Interfaces*.

logical-dbname

The logical name of the database on whose connection you want to check and set the user identity. The logical database name must be a character string enclosed in quotes, or a character expression. If you do not specify this argument, the compiler inserts the name of the database that is connected when the procedure is compiled. If you omit this argument and more than one database is connected, ABL raises an error.

Notes

- Within a transaction on a:
  - **Multi-tenant database** — Any attempt to set an identity for the connection that changes the current database tenancy raises a run-time error.
  - **Non-multi-tenant database** — As a best practice, Progress Software recommends that you **not** set a new identity for the connection.
- To assign a user identity while making a database connection, use the CONNECT statement.
- After a user identity is set for a database connection, the AVM uses that identity to determine if the user has permission to access tables and fields in that particular database.
- This function has the following restrictions:
  - It does not generate any audit events, such as for login and logout.
  - It can change the user identity for only one database connection at a time.
  - The domain of the user identity you want to set must be configured to use the authentication system, _oeusertable.

- The domain of the user identity you want to set must be enabled in the `_User` table of the connected database.

Note: When using an STS-enabled database, the `_oeusertable` authentication system is not required, and accounts do not have to be in the `_User` table.

- Under the following conditions, the SETUSERID function returns a value of FALSE and does not assign a user identity to the user:
  - There are no entries in the `_User` table.
  - There is no `_User` record with the same user ID as the one supplied with the SETUSERID function.
  - The password supplied with the SETUSERID function does not match the password in the `_User` table record of the specified user ID.
- When using the SETUSERID function, ABL returns a compiler error under the following conditions:
  - There is no database connected.
  - The *`logical-dbname`* argument is omitted, and more than one database is currently connected.
- When specifying the *`logical-dbname`* argument, you must provide the name of the logical database, not the physical database.
- SETUSERID encodes the *`password`* argument and then compares the result with the value stored in the `_User._password` field of the `_User` table.
- After SETUSERID returns a value of TRUE and assigns the authenticated user ID to a database connection:
  - ABL uses that user ID when the user compiles procedures.
  - Subsequent invocations of the USERID function for the same database connection return the assigned user ID.
- If the root user ID does not exist in the `_User` table, SETUSERID returns a value of FALSE when supplied with a userid of root. If the `_User` table does have a root entry, the user who assumes that user ID has all the privileges associated with the root user ID on UNIX.
- You must create and enable a blank user ID (`""`) if you want to set the user ID to a null value.
- See *OpenEdge Getting Started: Identity Management*, *OpenEdge Development: Programming Interfaces*, and *OpenEdge Data Management: Database Administration* for more information on user privileges.
- Once an initial database connection is established, you can also use the SET-CLIENT( ) method (on the SECURITY-POLICY system handle) or the SET-DB-CLIENT function to set the user identity for the connection.
- Any database connection whose user identity is set using this function locks out the SET-CLIENT( ) method from setting an identity for that connection. To unlock and allow the SET-CLIENT( ) method to set the identity for a locked database connection, call the SET-DB-CLIENT function for the connection, passing the Unknown value (`?`) for its *`client-principal-handle`* parameter. This unlocking of a database connection leaves the current connection identity unchanged.
- For each user identity authenticated by the SETUSERID function, the function creates a sealed security token containing the user credentials for the database connection, which you can return as a client-principal object using the GET-DB-CLIENT function. You can then use this sealed client-principal in a single sign-on (SSO) operation to set the identity of another database connection or ABL session.
- This function returns FALSE and with messages returned in the ERROR-STATUS system handle if:

- The format of the non-qualified user ID or domain name is invalid or the value has contains characters outside of the restricted user ID or domain name character sets (see the QUALIFIED-USER-ID attribute entry)
- Multiple databases are connected and *logical-dbname* is not passed or resolves to the Unknown value (?)
- Multiple databases are connected and *logical-dbname* does not refer to a connected database

- This function does not attempt set the connection identity for the foreign data source of a DataServer connection. However, it does attempt to set the connection identity for the OpenEdge schema holder database.

AUDIT-EVENT-CONTEXT attribute (enhanced for STS)

What's new

The AUDIT-EVENT-CONTEXT is now consistently stored in the `_Event-detail` field in the audit record.

Description

The audit event context for a client-principal object. The AVM stores this application-defined audit context in the `_Event-detail` field in the audit record created for an audit event generated during user authentication with the SEAL( ) method, SECURITY-POLICY:SET-CLIENT( ) method, SET-DB-CLIENT function, or AUTHENTICATION-FAILED( ) method, and with the LOGOUT( ) method. If not specified, the `_Event-detail` field in the audit record is left blank.

You can also use this value as an alternate index for querying the audit event record.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

If not already assigned, during any OpenEdge-performed user authentication operation, OpenEdge assigns the value of this attribute (if any) from the domain configuration used to authenticate the asserted identity before sealing the client-principal object.

Once the client-principal object is sealed, this attribute is read-only, and attempting to write to it raises a run-time error.

Also, attempting to set this attribute to a value longer than 200 characters raises a run-time error.

See also

SEAL( ) method (enhanced for STS) on page 58

LOGIN-STATE attribute (enhanced for STS)

What's new

This attribute now has five more possible values to support STS: NO-LOGIN, NO-ACCESS, REVOKED, DISABLED, and LOCKED.

Description

Returns a character value that represents the current state of the client-principal object.

Data type: CHARACTER

Access: Read-only

Applies to: Client-principal object handle

The following table lists the valid values and how they can be set.

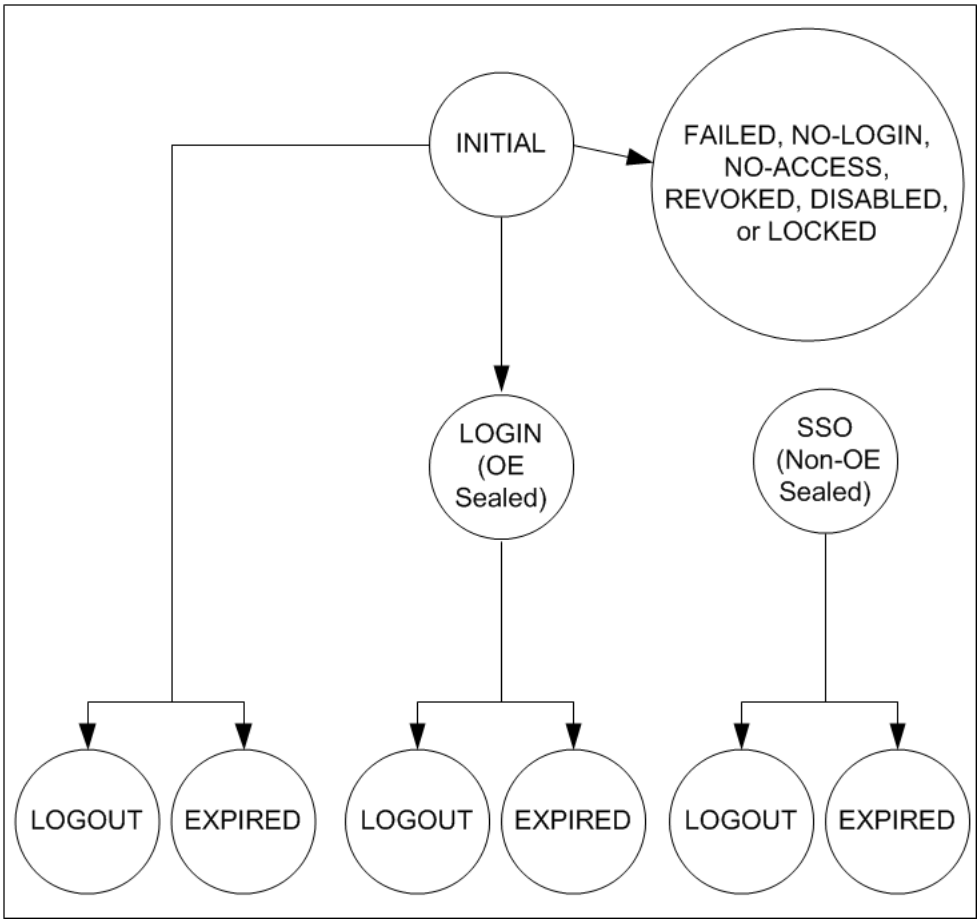
Table 1: Client-principal LOGIN-STATE attribute values

This value...	Is set...
"INITIAL"	(Default value) From the creation of an unsealed client-principal object using the CREATE CLIENT-PRINCIPAL until the object is sealed, and after calling the INITIALIZE( ) on a sealed or unsealed client-principal object in any state.
"LOGIN"	After the user identity represented by an unsealed client-principal object is successfully authenticated and the object is sealed using the SEAL( ) method (enhanced for STS) on page 58, SECURITY-POLICY:SET-CLIENT( ) method, the SET-DB-CLIENT function, or via an OpenEdge STS.
"SSO"	After a client principal is sealed by a non-OpenEdge user authentication process.
"LOGOUT"	After executing the LOGOUT( ) method on a sealed client-principal object in the LOGIN state or SSO state.
"EXPIRED"	After executing the SEAL( ) method, SET-CLIENT( ) method, SET-DB-CLIENT function, or IMPORT-PRINCIPAL( ) method, VALIDATE-SEAL( ) method (enhanced for STS) on page 60, or some external, non-OpenEdge seal procedure and the date and time set for the LOGIN-EXPIRATION-TIMESTAMP attribute has expired.
"FAILED"	After executing the AUTHENTICATION-FAILED( ) method on an unsealed client-principal object or after a user authentication operation fails on an unsealed client-principal object when calling the SET-CLIENT( ) method or SET-DB-CLIENT function.
"NO-LOGIN"	The login operation could not be completed. This state is set after a user authentication operation on an unsealed client-principal object fails due to some policy restriction when calling the SET-CLIENT( ) method or SET-DB-CLIENT function, where the authentication is done by an OpenEdge STS.
"NO-ACCESS"	The user is denied access. This state is set after a user authentication operation on an unsealed client-principal object fails due to some policy restriction when calling the SET-CLIENT( ) method or SET-DB-CLIENT function, where the authentication is done by an OpenEdge STS.
"REVOKED"	The user account's access has been revoked. This state is set after a user authentication operation on an unsealed client-principal object fails due to some policy restriction when calling the SET-CLIENT( ) method or SET-DB-CLIENT function, where the authentication is done by an OpenEdge STS.

This value...	Is set...
`"DISABLED"`	The user account is disabled. This state is set after a user authentication operation on an unsealed client-principal object fails due to some policy restriction when calling the SET-CLIENT( ) method or SET-DB-CLIENT function, where the authentication is done by an OpenEdge STS.
`"LOCKED"`	The user account is locked. This state is set after a user authentication operation on an unsealed client-principal object fails due to some policy restriction when calling the SET-CLIENT( ) method or SET-DB-CLIENT function, where the authentication is done by an OpenEdge STS.

The following figure illustrates the supported transitions between the states described in the table above.

Figure 1: Valid changes to LOGIN-STATE attribute value



When the AVM sets this attribute, it also sets the STATE-DETAIL attribute with a description of the current state. If this setting occurs as a result of invoking the AUTHENTICATION-FAILED( ), the AVM uses any string value you pass to this method to set the STATE-DETAIL attribute.

Attempting to write to this attribute returns a warning message.

SEAL( ) method (enhanced for STS)

What's new

After an STS-enabled database is connected to a given session, you cannot call the SEAL( ) method on the client principal with an encoded domain access code. After an STS-enabled database connection is made, you can only call SEAL( ) with a clear text domain access code.

Description

As part of a successful application-performed authentication operation, you can invoke this method on an unsealed client-principal in order to seal the object using the domain access code you pass as a parameter. Sealing the client-principal begins a login session to assert the identity it represents. You can then pass the sealed object as input to the SECURITY-POLICY:SET-CLIENT( ) method or the SET-DB-CLEINT function to set the identity for the current ABL session and one or more of its database connections using a single sign-on (SSO) operation.

Note: You only need to call the SEAL( ) method if your ABL application successfully performs its own authentication of the user's credentials and therefore needs to initialize and seal a client-principal to represent the authenticated user identity and begin a user login session. The domain that you use to initialize the client-principal is typically configured with either the built-in `_extsso` authentication system or a user-defined authentication system that is enabled for SSO-only operations to validate and set a user identity. For all other supported authentication systems, such as `_oeusertable`, `_oslocal`, or user-defined authentication systems that are enabled for user authentication using an ABL authentication plugin or if the authentication is done via STS, you can pass an appropriately initialized unsealed client-principal object directly as input to the SET-CLIENT( ) method or SET-DB-CLIENT function to authenticate the identity, seal the client-principal object, create the login session, and set the identity for the ABL session or database connection in a single user authentication operation.

You must validate and set values for the following attributes on the client-principal before you can seal the object with this method:

- USER-ID attribute
- DOMAIN-NAME attribute
- SESSION-ID attribute

The AVM raises a run-time error if:

- You do not appropriately set these attributes.
- The client-principal object is already sealed.
- An STS-enabled database is already connected to a session, and you invoke SEAL( ) with an encoded domain access code. This restriction prevents unauthorized access to the database using a client principal generated locally. After an STS-enabled database connection is made, you can only call SEAL( ) with a clear text domain access code.

You can seal a client-principal object only once per user login session. You can then use the VALIDATE-SEAL( ) method (enhanced for STS) on page 60 to validate the seal whenever necessary.

Once sealed, you can no longer set any attributes or user-defined properties for the object.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
SEAL ( domain-access-code )
```

domain-access-code

A case-sensitive character expression containing the access code that you have defined for the user's domain. The AVM uses this access code to generate the message authentication code (MAC) with which to seal the client-principal object. The AVM converts this access code to UTF-8 before using it, which ensures a consistent value regardless of code page settings. A matching domain with the specified access code must be registered in a trusted domain registry in order to successfully validate the identity that is sealed in this client-principal object.

Caution: The sealed client-principal created by this method is not usable if no trusted domain registry contains the same combination of domain name and access code. Also, note that given any client-principal that is sealed with it, this domain access code provides session and database access when validated using a single sign-on (SSO) operation against a trusted domain registry that contains a matching domain with the same access code. Therefore, Progress Software corporation **strongly** recommends that you take steps to hide or otherwise protect the domain access code in your ABL code from access (hacking) by unauthorized users.

If successful, this method returns TRUE. The method also sets the SEAL-TIMESTAMP attribute with the time stamp for when the client-principal object was sealed, and sets the LOGIN-STATE attribute (enhanced for STS) on page 55 to `"LOGIN"`. If not successful, the method returns FALSE.

The method also checks the LOGIN-EXPIRATION-TIMESTAMP attribute. If the client-principal object expires before you can seal it, the AVM sets the LOGIN-STATE attribute to `"EXPIRED"` and the method returns FALSE.

Calling this method generates a login audit event and creates an audit record for the event in all connected audit-enabled databases according to each database's current audit policy settings.

The following code fragment illustrates how to use the SEAL( ) method:

```
DEFINE VARIABLE hCP     AS HANDLE    NO-UNDO
DEFINE VARIABLE key     AS CHARACTER NO-UNDO.
DEFINE VARIABLE lResult AS LOGICAL   NO-UNDO.
. . .
CREATE CLIENT-PRINCIPAL hCp.
. . .
lResult = hCP:SEAL(key).
```

Note: You typically call this method on an unsealed client-principal object in the INITIAL state only as a result of a **successful** application-performed user authentication in order to seal the object in the LOGIN state. If you want to invalidate an unsealed client-principal object in the INITIAL state because the identity it represents has failed an application-performed user authentication operation, call the AUTHENTICATION-FAILED( ) method, which seals the client-principal in the FAILED state. If you want to invalidate, and terminate the login session for, a **sealed** client-principal object that is in the LOGIN state, call the LOGOUT( ) method, which sets the sealed client-principal to the LOGOUT state.

See also

LOGIN-STATE attribute (enhanced for STS) on page 55, VALIDATE-SEAL( ) method (enhanced for STS) on page 60

VALIDATE-SEAL( ) method (enhanced for STS)

What's new

VALIDATE-SEAL( ) now generates an audit event when the method finds the client principal to be expired for the first time. When this happens, the method generates an audit event for a logout operation.

Description

Validates the message authentication code (MAC) generated by the SEAL( ) method to seal a client-principal object.

You can use this method to validate the seal whenever necessary.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
VALIDATE-SEAL ( [ domain-access-code ] )
```

domain-access-code

An optional character expression containing the access code that is defined for the user's domain and that originally was used to seal the client-principal object. The AVM converts this value to a UTF-8 before using it, which ensures a consistent value regardless of code page settings.

ABL raises ERROR if the client-principal object is not:

- Sealed
- In the LOGIN state

If you specify a domain access code, the AVM uses the specified value to validate the seal.

Caution: Given any client-principal that is sealed with it, this domain access code provides session and database access when validated using a single sign-on (SSO) operation against a trusted domain registry that contains a matching domain with the same access code. Therefore, Progress Software corporation **strongly** recommends that you take steps to hide or otherwise protect any domain access code that you specify in your ABL code from access (hacking) by unauthorized users.

If you do not specify a domain access code, the AVM uses the access code defined for the domain in the trusted domain registry to validate the seal.

The AVM validates the seal by comparing it to the MAC generated from either the specified domain access code or the matching domain access code stored in the trusted domain registry. If the seal matches the generated MAC, the seal is valid and this method returns TRUE. Otherwise, the seal is invalid and this method returns FALSE.

The AVM also checks the LOGIN-EXPIRATION-TIMESTAMP attribute. If the client-principal object expires before you can validate its seal, the AVM sets the LOGIN-STATE attribute to "EXPIRED", and returns FALSE.

VALIDATE-SEAL( ) only generates an audit event when the method finds the client principal to be expired for the first time. When this happens, the method generates an audit event for a logout operation.

The following code fragment illustrates how to use the VALIDATE-SEAL( ) method:

```
DEFINE VARIABLE hCP    AS HANDLE    NO-UNDO.
DEFINE VARIABLE key    AS CHARACTER NO-UNDO.
DEFINE VARIABLE val-ok AS LOGICAL   NO-UNDO.
. . .
CREATE CLIENT-PRINCIPAL hCp.
. . .
val-ok = hCP:VALIDATE-SEAL(key).
```

See also

LOGIN-STATE attribute, SEAL( ) method in *OpenEdge Development: ABL Reference*

Progress.Security.PAMStatus class (enhanced for STS)

What's new

To support STS authentication, five new properties have been added to this class: AccessDenied, AccessRevoked, AccountDisabled, LoginDenied, and LoginLockout. See below for links to their entries.

Description

A class whose static members are the valid status codes that can be returned by an ABL authentication callback procedure, which can be called during the execution of an OpenEdge user authentication operation. See *OpenEdge Development: Programming Interfaces* for more information about authentication callback procedures.

Definition

This class is FINAL and cannot be inherited.

Serializable:

No

Constructors

This class only contains a private constructor.

Super Class

Progress.Lang.Object

Interfaces

This class does not implement interfaces (beyond those it inherits from its base class).

Public Properties

AccessDenied property on page 62	AccessRevoked property on page 62	AccountDisabled property on page 62
AccountExpired property on page 63	AuthenticationAccess property on page 63	AuthenticationFailed property on page 63

Custom property on page 63	InvalidConfiguration property on page 63	LoginDenied property on page 64
LoginLockout property on page 64	MaxTries property on page 64	MissingCredentials property on page 64
PasswordExpired property on page 64	PermissionDenied property on page 65	Success property on page 65
UnknownUser property on page 65	–	–

Public Methods

This class does not contain methods (beyond those it inherits from its base class).

Public Events

This class does not contain events.

Notes

- An authentication callback procedure must return one of the values specified on the above table, otherwise the AVM raises a standard authentication failure error (`PAMStatus:AuthenticationFailed`).
- The values associated with each property of this class conform to Identity Management (IdM) standards. The AVM maps them to the corresponding values defined by the RFC 86 PAM (Pluggable Authentication Modules) API specification.

AccessDenied property

An authentication status code that indicates authentication failure because of a denied access attempt.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

AccessRevoked property

An authentication status code that indicates authentication failure because access has been revoked.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

AccountDisabled property

An authentication status code that indicates authentication failed because the user account or domain has been disabled.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

AccountExpired property

An authentication status code that indicates authentication failed because the user account has expired.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

AuthenticationAccess property

An authentication status code that indicates authentication failed because user account information is not accessible.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

AuthenticationFailed property

An authentication status code that indicates a non-specific authentication failure, which may be the result of a user account being disabled or some other user login restriction.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

Custom property

An authentication status code that indicates a custom error condition that is defined and returned by an ABL callback procedure.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

InvalidConfiguration property

An authentication status code that indicates authentication failure due to a missing or invalid OpenEdge domain configuration.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

LoginDenied property

An authentication status code that indicates authentication failed because of a denied login attempt.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

LoginLockout property

An authentication status code that indicates authentication failed because of a login failure lockout (e.g., too many failed attempts).

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

MaxTries property

An authentication status code that indicates failure because the limit on the number of login retries was exceeded.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

MissingCredentials property

An authentication status code that indicates failure because one or more of the required user credentials is missing from the CLIENT-PRINCIPAL.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

PasswordExpired property

An authentication status code that indicates authentication failed because the user account password has expired.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

PermissionDenied property

An authentication status code that indicates failure because user does not have the necessary privileges to access the user account information required to complete an authentication or SSO operation.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

Success property

An authentication status code that indicates that user credentials have been successfully authenticated.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

UnknownUser property

An authentication status code that indicates that the user account does not exist in the user account system defined in the OpenEdge domain.

Data type: INTEGER

Access: PUBLIC STATIC Read-only

Applies to: Progress.Security.PAMStatus class (see *OpenEdge Development: ABL Reference*)

Startup parameters

The sections that follow contain new startup parameters.

Certificate store path (-certstorepath)

Use Certificate store path (`-certstorepath`) to specify the directory location of the Certificate Authority (CA) certificates to validate an SSL/TLS enabled server's certificate.

Operating system and syntax	**UNIX / Windows**	**-certstorepath *directory-name***		
Use with	**Maximum value**	**Minimum value**	**Single-user default**	**Multi-user default**
Client Connection, Authentication Gateway STS	—	—	`%DLC%/certs`	`%DLC%/certs`

directory-name

Specify the directory where your Certificate Authority (CA) certificates are located.

The database server and the Authentication Gateway STS server share the Certificate store path (-certstorepath) setting.

Domain (-domain) (for STS capability)

For use with the OpenEdge Authentication Gateway only. Use Domain (-domain) with OS User (-OSUser) to perform a single sign-on (SSO) connection to an STS-enabled database using the OS credentials.

Operating system and syntax	UNIX / Windows	-domain *domain-name*		
Use with	**Maximum value**	**Minimum value**	**Single-user default**	**Multi-user default**
Client Connection	—	—	—	—

domain-name

The name of the domain to be used with the OS credentials to enable an SSO connection.

Domain (-domain) can only be used with OS User (-OSUser); using -domain on its own raises an error. If -domain and -U are both used, you will get a warning, and the domain in -U will be used instead of what's specified by -domain. For more information, see OS User (-OSUser) (for STS capability) on page 68 for more information on how these parameters work together.

Key store path (-keystorepath)

Use Key store path (-keystorepath) to specify the directory location of an SSL/TLS enabled server's private key and certificate.

Operating system and syntax	UNIX / Windows	-keystorepath *directory-name*		
Use with	**Maximum value**	**Minimum value**	**Single-user default**	**Multi-user default**
Client Connection,Database Server, Authentication Gateway STS	—	—	%DLC%/keys	%DLC%/keys

directory-name

Specify the directory where your key store is located.

The database server and the Authentication Gateway STS server share the Key store path (-keystorepath) setting.

No Host Verify (-nohostverify) (for STS capability)

Use No Host Verify (`-nohostverify`) to turn off host verification for a Secure Sockets Layer (SSL) connection to a database server or to the OpenEdge Authentication Gateway server.

Operating system and syntax	UNIX / Windows	`-nohostverify`		
Use with	**Maximum value**	**Minimum value**	**Single-user default**	**Multi-user default**
Client Connection, Authentication Gateway server	—	—	—	—

Without this parameter specified, the client compares the host name specified in the connection with the Common Name specified in the server certificate, and raises an error if they do not match. With this parameter specified, the client never raises the error.

While your database connection may or may not be running with SSL enabled for communication to its remote clients, the communication to the OpenEdge Authentication Gateway server is always run with SSL enabled.

If the database is running with SSL enabled, No Host Verify (`-nohostverify`) is set the same for both the OpenEdge Server SSL communication as well as for the OpenEdge Authentication Gateway server communication.

For more information, see *OpenEdge Getting Started: Core Business Services - Security and Auditing*.

No Session Reuse (-nosessionreuse) (for STS capability)

Use No Session Reuse (`-nosessionreuse`) when you do not want to reuse a Secure Sockets Layer (SSL) session ID to reconnect to the same SSL-enabled database server or to reconnect to the OpenEdge Authentication Gateway STS server.

Operating system and syntax	UNIX / Windows	`-nosessionreuse`		
Use with	**Maximum value**	**Minimum value**	**Single-user default**	**Multi-user default**
Client Connection, Authentication Gateway STS	—	—	—	—

While your database connection may or may not be running with SSL enabled for communication to its remote clients, the communication to the Authentication Gateway STS Server is always run with SSL enabled.

If the database is running with SSL enabled, No Session Reuse (`-nosessionreuse`) is set the same for both the OpenEdge Server SSL communication as well as for the STS Server communication.

For more information, see *OpenEdge Getting Started: Core Business Services - Security and Auditing*.

Note: OpenEdge SSL turns on SSL session reuse by default. So, after the initial connection to a given host (`-H`) and port (`-S`), each subsequent connection to the same host and port restarts the SSL session and ignores any different connection parameters that are specified for the subsequent connection, including `-nosessionreuse`. If you want to change SSL socket options (such as `-nohostverify`) for each subsequent connection to a given host and port, be sure to specify the `-nosessionreuse` parameter on the initial SSL socket connection to that same host and port.

OS User (-OSUser) (for STS capability)

For use with the OpenEdge Authentication Gateway only. Use OS User (`-OSUser`), with or without the Domain (-domain) (for STS capability) on page 66 connection parameter, to enable a single sign-on (SSO) connection to an STS-enabled database using the OS credentials.

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

When authentication is done via a client and you do not specify `-U`/`-P` at connection time, the client connects to the database using the blank user id or performs an SSO connection using the OS credentials. This method of authentication does not work with an STS-enabled database; instead, use `-OSUser` to generate an SSO token using the OS credentials, which will then be exchanged for a login token by the STS.

There are some additional requirements and considerations to be aware of when using `-OSUser`:

- The domain configuration must be properly set up to allow SSO token exchange. See Configuring domains on page 25 for more information.
- When `-OSUser` is used alone (without the Domain (-domain) (for STS capability) on page 66 parameter), the STS authenticates using the OS user credentials and the blank domain. Note that you must have the blank domain set up on the STS to allow the token exchange.

  Caution: Use of the blank domain is not a recommended practice, particularly in multi-domain environments.

- You can specify both `-OSUser` and `-U`, and if the user ID given by `-U` is fully qualified, the domain from that user ID is used. However, the user name specified by `-U` must match the OS user name exactly.
- You can use `-OSUser` and `-domain` to use the OS user credentials and a specified domain to make an SSO connection. The domain specified by `-domain` is used instead of the default, unless `-U` is also used and contains a fully qualified user ID. See the Domain (-domain) (for STS capability) on page 66 parameter entry for more information.

The following table summarizes the combinations of parameters you can use for an SSO connection to an STS-enabled database:

Connection parameters	Credentials used for SSO connection
`-OSUser`	OS user ID, blank domain

Connection parameters	Credentials used for SSO connection
-OSUser -U *user[@domain]*	OS user ID (which must match *user*), *domain* if specified, otherwise blank domain
-OSUser -domain *domain-name*	OS user ID, *domain-name*

Security cache size (-secsize)

Operating system and syntax	UNIX / Windows	**-secsize *n***		
Use with	**Maximum value**	**Minimum value**	**Single-user default**	**Multi-user default**
DBS	32768	0	-	512

n

The size, in number of cache entries, of the primary and secondary security caches.

For best performance, you should set Security cache size (-secsize) large enough that all your existing records fit in the primary cache. Pick the value for -secsize based on available shared memory and the size of the _sec-authentication-domain table. Each entry consumes approximately 450 bytes. For example, if -secsize is 512, roughly 512 * 450 = 0.25 Mb is allocated for the primary cache, and also the secondary cache, for a total of 0.5 Mb. If memory is not an issue, pick a value close the number of records in _sec-authentication-domain.

If you set the value between 1 and 12, it is automatically increased to 13.

You can set Security cache size (-secsize) to 0 (zero) if you never intend to enable use of the security cache on your database.

For use with the OpenEdge Authentication Gateway only, use Security Cache Size (-secsize) to specify the size (in number of security cache entries) of the security cache. The security cache holds fields from the _sec-authentication-domain and _Tenant-Name tables in memory, eliminating the need to read those values from the database during connection validation.

At startup, if your database is currently enabled for authentication through use of the OpenEdge Authentication Gateway, two security caches are allocated, a primary cache and a secondary cache.

Existing fields are cached in the primary security cache and are preloaded by a broker. After instantiation, no new entries are ever added to the primary cache. If the primary cache is filled, remaining entries are inserted into the secondary cache. All new entries are automatically cached in the secondary cache. During normal database activity, fifty percent (50%) of the cache entries must be evictable. If the number of fixed cache entries is more than 50% of all entries, new entries cannot be added to the cache and creation of a new _sec-authentication-domain record fails. If this occurs, an informational message is written to the database log file, as shown:

```
Please increase security cache size using database startup parameter -secsize.
Cannot perform this operations because the database security cache is too small.
```

STS logging level (-stsloggінglevel)

Operating system and syntax	UNIX / Windows	`-stsloggінglevel n`		
Use with	**Maximum value**	**Minimum value**	**Single-user default**	**Multi-user default**
DBS,CC	5	0	0	0

n

The logging level:

- **0** — none
- **1** — fatal errors only
- **2** — all errors
- **3** — all errors and warnings
- **4** — all errors, warnings, and informational messages
- **5** — all errors, warnings, informational, and debug messages
- Values less than 0 generate an error; values greater than 5 are reset to 5 internally

For use with the OpenEdge Authentication Gateway only, use STS logging level (`-stsloggінglevel`) to specify the logging level value for STS client interface logging. This client-side startup parameter can aid you in debugging authentication problems in an OpenEdge Authentication Gateway deployment.

Notes

- This is a per database connection setting. If you connect to more than one database, each database can have a different `-stsloggінglevel` value.
- All output generated by `-stsloggінglevel` is written to the database `.lg` file.
- If the primary broker specifies `-stsloggінglevel` at startup, the shared default is changed to the broker's value. Automatic servers and secondary brokers can only use the shared default. This parameter is not listed as a database startup parameter in the `.lg` file, VSTs or promon.
- Automatic servers and secondary brokers can only use the shared default value.
- Single user connections, self service clients and manual servers use the default shared value by default, but can override the use of the default by specifying their own `-stsloggінglevel` value upon connection.
- Remote clients use their database servers' setting according to the above rules.

Database utilities

The sections that follow contain new database utilities.

PROUTIL DISABLEAUTHGATEWAY qualifier

Disables the OpenEdge Authentication Gateway for a database.

Syntax

```
proutil db-name -C disableauthgateway
```

Parameters

db-name

Name of the database to have use of the OpenEdge Authentication Gateway disabled.

DISABLEAUTHGATEWAY removes the `_db-option` record with `_db-option-code="_db.sts.url"` and `_db-option-type=3`. (These are the values set by using the STS URL Utility prior to enabling the database to use the Authentication Gateway.) After disabling use of the OpenEdge Authentication Gateway, connection role enforcement no longer takes place. If the database is enabled for connection role enforcement at the time DISABLEAUTHGATEWAY executes, the following warning message is reported:

```
Warning: Disabling the OpenEdge Authentication Gateway feature disables
enforcement of connection role security.
```

After successfully executing DISABLEAUTHGATEWAY, the following message is reported:

```
The Authentication Gateway has been disabled for database dbname.
```

If auditing is enabled on your database, Audit event 11902 is recorded.

PROUTIL ENABLEAUTHGATEWAY qualifier

Enables the OpenEdge Authentication Gateway for a database.

Syntax

```
proutil db-name -C enableauthgateway
```

Parameters

db-name

Name of the database to be enabled for use of the OpenEdge Authentication Gateway.

PROUTIL ENABLEAUTHGATEWAY enables an OpenEdge database to use the OpenEdge Authentication Gateway for authentication.

Enabling the database validates that there is at least one valid enabled domain to avoid lockout of all logins. If there is no valid domain, ENABLEUATHGATEWAY fails with the following error:

```
Enable failed due to lack of a valid enabled domain.
```

Additionally, there must a valid URL specified in the `_db-option` table. Set the URL with the STS URL Utility. For details, see the STS URL Utility in *OpenEdge Getting Started: Authentication Gateway Guide*. If the base URL name is not set correctly, enablement fails with the following error:

```
Enable failed due to lack of a valid STS URL.
```

For the most secure deployment, Progress Software Corporation recommends that connection role enforcement is also enabled on the database. If the database is NOT enabled for connection role enforcement when enabling the OpenEdge Authentication Gateway, then the following warning is reported:

```
Warning: It is recommended that connection role security enforcement be configured
when using the OpenEdge Authentication Gateway.
```

When successfully enabled, ENABLEAUTHGATEWAY reports:

```
Authentication Validation has been enabled for the database dbname
```

If auditing is enabled, Audit event 11901 is recorded.

Auditing events for OpenEdge Authentication Gateway

The following table lists audit events added in support of the OpenEdge Authentication Gateway.

Table 2: Authentication Gateway audit events

Event Id	Event Name	Event Type	Event Description	Policy Name
10602	_pvm.db.connect.fail	User	Database connect failed	PSC-User
10612[1]	_sql.db.connect.fail	User	SQL database connect failed	PSC-User
11901	_sys.db.sts. authgatewayenable	Utility	Enable Authentication Gateway	PSC-DB-Admin
11902	_sys.db.sts. authgatewaydisable	Utility	Disable Authentication Gateway	PSC-DB-Admin

Audit records for connection attempts now contain more information. This applies to connection audit events _pvm.db.connect (10600) and _pvm.db.connect.fail (10602). The audit `_Event-detail` field stores more information for these events, as shown below:

```
OS-username:process-id:OS-type:
ip-addr:OE-client-type:db-conn-type:
OS-userid-or-GUID
```

Note that:

- *`OS-username`* will be in the form of Windows-domain\username on Windows.
- *`OS-type`* will be a string indicating the platform, e.g., Linux, Solaris, Windows, etc.

[1] Connecting SQL via the Authentication Gateway is not supported in Release 11.6.2.

- *OE-client-type* will be a string to record if the process was a batch client, an ABL client, a PAS for OpenEdge instance, or a Web (WebHandler) client.
- *db-conn-type* will be single-user, self-service, or remote.
- *OS-userid-or-GUID* is either the numeric value for the userid on UNIX, or a GUID representing the SID (security Identifier) on Windows. Both values come from the OS.

OpenEdge Authentication Gateway utilities

Change the keystore password utility (changeP12pwd)

This utility changes the keystore password in a specified keystore file. The utility is found in the `/conf` folder of a PAS for OpenEdge instance.

The default password is `password` in the default keystore file: `%DLC%/bin/tomcat-keystore.p12`. Obviously, the password should be changed when you are deploying a production instance of PAS for OpenEdge.

Note:

You can check the current password by running: `tcman config psc.as.https.keypass`. After changing the password with `changeP12pwd`, set `psc.as.https.keypass` to the new password. (See the example below.)

Syntax

```
changeP12pw { -keystorefile file_name -keypassword
current_password } | -h
```

Parameters

-file_name

Specifies the file name (or complete path) of the keystore file.

-current_password

Specifies the existing password in the keystore file.

-h

Displays a list of the `changeP12pwd` parameters.

Example

The following example changes the default password. The `changeP12pwd` utility is being invoked from the `/conf` directory on the default keystore file that is in the same directory.

```
proenv>changeP12pwd  -keystorefile  tomcat-keystore.p12 -keypassword password
MAC verified OK
##### Take backup of original tomcat-keystore.p12 as tomcat-keystore_bk.p12
#####.alias ############## : test2
Loading 'screen' into random state - done
Enter Export Password: newpassword
Verifying - Enter Export Password:  newpassword
"### Success: tomcat-keystore.p12 password changed ###"
```

Notice that:

- The utility creates a backup of the original keystore file. In this case, it is `tomcat-keystore_bk.p12`.
- The utility prompts you to enter the new password and to verify the entry.
- The utility indicates if the change was successful.

After a successful password change, reset `psc.as.https.keypass` to the new value:

```
tcman config psc.as.https.keypass=newpassword
```

Data export utility

This utility is an ABL procedure that provides an administrator with a tool that can output Domain records from an OpenEdge database. This output can be used as input for an STS application's Domain configuration file and for the Domain keystore file generator (`gendomreg`).

The procedure writes two log files, which are created in the directory specified by the `FOLDER` parameter:

- `domain_dump_runner.log` — a log file for the procedure (independent of the data export)
- `domain_dump.log` — a log file for the actual data export

Syntax

```
_progres -p "OpenEdge/DataAdmin/Util/dump_domains.p"
        -db db-name [ -db db-name [ . . . ]
] -param "PARAM_NAME:VALUE  [,PARAM_NAME:
VALUE  [,. . .]  ]"
```

Parameters

The following table contains a description of the supported parameters and their valid values:

Parameter	Valid values	Notes
`LOG`	Any of the names in `OpenEdge.Core.LogLevelEnum`	If no value is passed in, or the value is invalid (no in the enum), the WARN level is used.

Parameter	Valid values	Notes
FOLDER	An existing folder into which data is exported and log files are written	If no value is specified, or if the value is invalid (i.e. the folder does not exist), the WRKDIR environment variable is queried and if not found, the SESSION:TEMP-DIR is used
MAKESSO	A logical TRUE/FALSE value	The ABL LOGICAL( ) function is used to evaluate the value. The default is FALSE if no value is specified or if the value doesn't resolve to true or false.
CONFIG-FILE	A filename that will be created	Used for the domain configuration file. Sets the export utility's DomainConfigFile property.
KEYSTORE-FILE	A filename that will be created	Used for the domain keystore. Sets the export utility's DomainKeystoreFile property.

STS Client Utility (stsclientutil)

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

- Establish the OpenEdge installation's ability to connect to a PAS for OE server and use its STS application (i.e. *ping* the STS).
- Test the ability of the OpenEdge installation to use an STS application for direct user login operation for a specific OpenEdge Domain (i.e. *authenticate* the user).
- Test the ability of the OpenEdge installation to use an STS application for operating system user login SSO to a specific OpenEdge Domain (i.e. *exchange* the OS security token for an OpenEdge security token).

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

-cmd *command*

Specify the task to perform from one of the following:

- **ping** — Send an empty message to an STS application to test HTTPS connections and STS Key features.
- **authenticate** — Send user login credentials to the STS application to test its Domain configuration.

  With authenticate, you must also specify the OpenEdge user-id used to send to the STS application for direct user login testing, as shown:

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

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

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

`-domain` *`domain`*

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

`-url` *`sts-url`*

Specify the URL of the STS application (secure PAS for OE server that includes a Security-Token-Service (STS) application) used by the OpenEdge database. There is no default for *`sts-url`*, the URL must be specified.

`command-options`

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

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

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

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

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

`-installpath` *`dlc-path`*

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

`-keystorepath` *`ks-path`*

Specify the operating system file path to where the STS Client Key is installed. If not specified, the default is `$DLC/keys`.

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

`-certstorepath` *`cs-path`*

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

`-logginglevel` *`level`*

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

`-nohostverify`

Specify this option to suppress the validation of the PAS for OpenEdge certificate's subject name against the URL host DNS name.

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

`-servername`

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

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

Examples

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

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

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

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

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

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

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

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

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

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

STS Key Utility (stskeyutil)

The `stskeyutil` command provides you with the capability to create and manage server and client keys. Major options include:

- `create`
- `verify`
- `install`
- `keyfile`

Testing options include:

- `keytest`
- `headertest`
- `ssotest`

The syntax of the specific commands is discussed in the sections that follow.

stskeyutil common options

All of the `stskeyutil` commands support the following common options:

```
-file filename
-keystorepath path
-silent
-url sts-url
-v
```

`-file` *`filename`*

Specify the STS server key file name. The default is `oests-key.ecp`.

`-keystorepath` *`path`*

Identify the directory location of the client file. The default is `%DLC%/keys`.

`-silent`

Do not prompt for password, access-codes, or file paths.

`-url` *`sts-url`*

Specify the STS server's url in the format: `https://<host>[:port]/[web-app]`.

`-v`

Provide additional run-time information as the utility executes (verbose).

Common options are represented by **[** *`common-options`* **]** in the utility syntax box.

stskeyutil create utility

Generates an STS Server Key file.

```
stskeyutil create
     [ -overwrite
     | -header head
     | -user usr
     | -roles roles
     | -domain name
     | -expires n
     ]
     [ common-options ]
```

`-overwrite`

Overwrite and existing key file. By default, an existing key file is not overwritten.

`-url` *`url`*

This is one of the common options that specifies the OpenEdge Authentication Gateway server's URL into the key. If not specified, no URL is written to the key. This does not affect the ability to use the key.

`-header` *`head`*

Specify a client STS Client Key in *`head`* to send to the to Authentication Gateway Server in the header. If not specified, the client uses the default header name (x-oests-token).

`-user` *`usr`*

Specify the Client-Prinicipal user-id in *`usr`*. If not specified, the default user-id is `oeclient`.

`-roles` *`roles`*

Assign roles for authorization to the OpenEdge STS resources. If not specified, the default role is `OESTSClient`.

`-expires` *`n`*

The amount of time before the key expires. Specify the number of days in *`n`*.

The `stskeyutil create` utility generates a sealed key file containing a sealed OpenEdge Client-Principal, using an administrator-supplied Domain access code. The Client-Principal field contents generated for the STS Server Key is not controllable by the administrator.

Note: When the OpenEdge STS server is running in a clustered environment, the administrator does not need to create an STS Server Key for each node. Assuming all nodes in the cluster share the same OpenEdge STS Server configuration, they share the STS Server Key. Note that when installing the STS Client Key, you can specify a different node name via the `-node` parameter.

---

stskeyutil verify utility

Verifies access to and the integrity of an STS Server Key file.

```
stskeyutil verify
    [ -seal | -info ]
    [ common-options ]
```

`-seal`

Specify to validate the STS server key's Domain Access Code.

`-info`

Specify to display server key information.

The `stskeyutil verify` utility password decrypts the *STS Server Key* file using an administrator supplied password and verifies its contents. (You can optionally verify the domains access code as well via the -seal parameter.) Successful verification yields the following message:

```
Server key validation success
```

Note: Adding `-info` to verbose tracing outputs certain key information fields for verification purposes. However, certain elements of the key file are not visible to the user so that the key cannot be fully generated from the information displayed.

stskeyutil install utility

Creates a new STS Client Key file from an STS Server Key file.

```
stskeyutil install -url sts-url
    [ -overwrite ]
    | -node node-name]
    [ common-options ]
```

`-url` *`sts-url`*

Specify the STS server's url in the format: `https://<host>[:port]/[web-app].`

`-overwrite`

Overwrite an existing key file. By default, an existing key file is not overwritten.

`-node` *`node-name`*

Specify the cluster node named *`node-name`*.

The `stskeyutil install` creates and installs an STS Client Key that corresponds to a specific OpenEdge STS server key. The resulting STS Client Key is encrypted and stored inside the OpenEdge installation's (default) `%DLC%/keys` directory.

Note: Each OpenEdge STS server used by an OpenEdge installation's products requires that a separate STS Client Key be created and installed using the STS Server Key. This requires the administrator to create the key by running `stskeyutil install` on the host where the STS Client Key file will be written and from within the OpenEdge installation the STS Client Key is generated for.

Each installed STS Client Key file is bound to a single OpenEdge installation, on a single server, and to the OpenEdge STS (URL) used for access. The URL must match exactly how it is set up on the database configuration. This binding is reflected by the hashed file name of the STS Client Key file.

When the OpenEdge STS server is running in a clustered environment it may not always be possible to generate each node's STS Client Key by logging directly into each node and executing the install command. To enable generating a STS Client Key for each cluster node, an administrator may follow this process:

- Ensure that each cluster node will use the same exact absolute path to the OpenEdge installation.

  Warning: If this is not true, stop now!

- Obtain each cluster node's node-name (found using the UNIX `uname -n`)
- Log in to the active cluster node as the OpenEdge administrator, and execute `stskeyutil install` for the current cluster node
- Repeat, executing `stskeyutil install` once for each additional cluster node, adding `-node` *`node-name`* to the command line

stskeyutil keyfile utility

Display the full path to a STS client key file.

```
stskeyutil keyfile -url sts-url
       [ -node node-name ]
       [ common-options ]
```

`-url` *`sts-url`*

Specify the STS server's url in the format: `https://<host>[:port]/[web-app]`.

`-node` *`node-name`*

Display the *STS Client Key* filepath for the cluster node named *`node-name`*.

The `stskeyutil keyfile` displays the full path to the STS client key file based on the URL of the OpenEdge STS Server (the default, or one you supply on the command line). Output to `stdout` shows the file path information.

Note:

When the OpenEdge STS server is running in a clustered environment, it may not always be possible to log into each node in order to obtain it's STS Client Key file name by executing the keyfile command. To enable discovering each cluster node's STS Client Key filename, follow this process:

- Obtain each cluster node's node-name (found using the UNIX `uname -n`)
- Log in to the active cluster node as the OpenEdge administrator
- Execute `stskeyutil keyfile` for the current cluster node
- Repeat, executing `stskeyutil keyfile` once for each additional cluster node adding `-node` *`node-name`* to the command line

stskeyutil keytest utility

Tests access to an existing STS Client Key file.

```
stskeyutil keytest -url sts-url
    [ -info | -seal -file name ]
    [ -node node-name ]
    [ common-options ]
```

`-url` *`sts-url`*

Specify the STS server's url in the format: `https://<host>[:port]/[web-app]`.

`-info`

Display client key information.

`-seal -file` *`name`*

Validate the client STS key's domain access code, by supplying the server STS key file in *`name`*.

`-node` *`node-name`*

Display the *STS Client Key* filepath for the cluster node named *`node-name`*.

An administrator may test an OpenEdge installation's ability to decrypt and use an STS Client Key file using `stskeyutil keytest`. Run this utility, entering the OpenEdge STS server's URL on the command line. Output to `stdout` displays success or failure information.

When the OpenEdge STS server is running in a clustered environment it may not always be possible to test each node's STS Client Key by logging directly into each node and executing `stskeyutil keytest`. To enable testing a STS Client Key for each cluster node an administrator may follow this process:

- Obtain each cluster node's node-name (found using the UNIX `uname -n`)
- Log in to the active cluster node as the OpenEdge administrator, and execute`stskeyutil keytest` for the current cluster node
- Execute `stskeyutil keytest` once for each additional cluster node, adding `-node` *`node-name`* to the command line

stskeyutil headertest utility

Test the use of a STS Client Key to produce an HTTP header for use in gaining authorization to an OpenEdge STS server.

```
stskeyutil headertest -url url
    [ -info ]
    [ common-options ]
```

`-url` *url*

Specifies the OpenEdge Authentication Gateway server's URL into the key. If not specified, no URL is written to the key.

`-info`

Display client key information.

Note: To perform`stskeyutil keytest` in a cluster's environment, it must be executed in the node the OpenEdge database and/or ABL client executes in. That is, the `-node` parameter is not supported.

---

stskeyutil ssotest utility

Test the use of an STS Client Key to produce an OS user-id token that may be validated by an OpenEdge STS server and exchanged for a full OpenEdge Client-Principal token.

Note: To perform `stskeyutil headertest` in a cluster's environment, it must be executed in the node the OpenEdge database and/or ABL client executes in. That is, the `-node` parameter is not supported.

```
stskeyutil ssotest -url url
    [ common-options ]
```

`-url` *url*

Specifies the OpenEdge Authentication Gateway server's URL into the key..

STS Connection Role Utility (stsconnroleutil)

As part of using the OpenEdge Authentication Gateway, database connection role authorisation is added. This allows DBA's to restrict the set of users who may connect to a database at any given time.The feature is enabled via a record in the `_db-options` table of the database that is checked when the use of the gateway is enabled. A default role called `_db.connection` is used to grant individual users access to the database. No wildcard grants are used.

The stsconnroleutil utility provides a DBA the ability to control which user accounts are granted membership in a database's connection role from the operating system command line. The connection role is used to grant/revoke an individual user's right to either connect to their database, or be set as the connection's current user. Options include:

- status
- enable
- disable
- list
- grantuser
- grantfile
- revokeuser
- revokefile

The syntax of the specific commands is discussed in the sections that follow.

stsconnroleutil status utility

Display the status connection role for the specified database.

```
stsconnroleutil status
    [ -folder folder-name]
    -db dbname [db-options]
        [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-folder` *folder-name*

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

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

`-db dbname [` *db-options* `]`

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

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

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

The `stsconnroleutil status` utility writes the status of the database connection roles to the file, `dbconnection_status.txt`. One line per database is written to the file in the format:

```
DbName,Enabled,RoleName
```

stsconnroleutil enable utility

Enables the *connection role* feature in each specified database.

```
stsconnroleutil enable
    [ -admin admin-username]
    [ -folder folder-name]
    -db dbname [db-options]
        [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-admin` *admin-username*

Specify the administrator username. If not specified, the current user is assumed to be the administrator.

`-folder` *folder-name*

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

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

`-db` *dbname* **[** *db-options* **]**

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

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

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

stsconnroleutil disable utility

Disables the *connection role* feature in each specified database.

```
stsconnroleutil disable
    [ -folder folder-name]
    -db dbname [db-options]
        [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-folder` *folder-name*

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

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

`-db` *dbname* **[** *db-options* **]**

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

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

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

stsconnroleutil list utility

List the current user's granted *connection role* membership for each specified database.

```
stsconnroleutil list
    [ -folder folder-name]
    -db dbname [db-options]
        [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-folder` *folder-name*

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

- The directory specified by the `WRKDIR` environment variable

- Current working directory
- The session temp-dir specified using `-T`

`-db dbname [ db-options ]`

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

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

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

The `stsconnroleutil list` utility writes the database connection roles to the file, `dbconnection_list.txt`. One line per grant is written to the file in the format:

`DbName,Grantee,CanGrant,Grantor,RoleName.`

stsconnroleutil grantuser utility

Grants the specified user membership in the connection role of each specified database.

```
stsconnroleutil grantuser
     -user username -can { yes | no }
    [ -folder folder-name]
    -db dbname [db-options]
         [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-grantee username`

Specify the user to grant connection role membership

`-can { yes | no }`

Specify if the user being granted connection role membership can grant membership to other users. You can also specify `true` or `false`.

`-folder folder-name`

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

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

`-db` *dbname* **[** *db-options* **]**

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

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

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

stsconnroleutil grantfile utility

Grants each user in the specified file membership in the connection role of each specified database.

```
stsconnroleutil grantfile -file filename
    [ -folder folder-name]
    -db dbname [db-options]
        [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-file` *filename*

Specify a file containing qualified user-ids to grant connection role membership. The format of the file is as follows:

- One user-id per line, with comma-delimited values
- The first field is the user-id
- The second field is the can-grant value (yes or no), indicating if the user can grant connection role membership to others.

`-folder` *folder-name*

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

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

`-db dbname` **[** *db-options* **]**

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

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

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

stsconnroleutil revokefile utility

Revokes membership in the connection role for each user in the specified file of each specified database.

```
stsconnroleutil revokefile -file filename
    [ -folder folder-name]
    -db dbname [db-options]
        [-db dbname [db-options] ... ]
    [abl-startup-options]
```

-file *filename*

Specify a file containing qualified user-ids to revoke connection role membership. The format of the file is one user-id per line.

-folder *folder-name*

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

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

-db *dbname* [*db-options*]

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

[-db *dbname* [*db-options*] ...]

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

stsconnroleutil revokeuser utility

Revokes membership in the connection role for the specified user of each specified database.

```
stsconnroleutil revokeuser
     -user username
    [ -folder folder-name]
    -db dbname [db-options]
         [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-grantee username`

Specify the user to revoke connection role membership

`-folder folder-name`

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

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

`-db dbname [ db-options ]`

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

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

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

STS URL Utility

The stsurlutil utility provides you with the capability to control which OpenEdge Authentication Gateway server a database will use to authenticate users during a connection or switch-user. Options include:

- status
- list
- update

As part of using the OpenEdge Authentication Gateway, database administrators need to provide an HTTPS URL. This value is held in a record in the _db-options table that is checked when the database is enabled to operate with the OpenEdge Authentication Gateway. Only valid HTTPS URLs are allowed, and none that point directly to the localhost. The syntax of the specific commands is discussed in the sections that follow.

stsurlutil list utility

List the current OpenEdge Authentication Gateway server URL for each specified database.

```
stsurlutil list
    [ -folder folder-name]
    -db dbname [db-options]
        [-db dbname [db-options] ... ]
    [abl-startup-options]
```

-folder *folder-name*

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

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

-db dbname [*db-options*]

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

[-db *dbname* [*db-options*] ...]

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

The stskeyutil list utility writes the gateway URL to the file, dbauthgateway_list.txt. One line per database is written to the file in the format:

```
DbName,URL
```

stsurlutil status utility

Display the status of the OpenEdge Authentication Gateway for each specified database.

```
stsurlutil status
    [ -folder folder-name]
    -db dbname [db-options]
        [-db dbname [db-options] ... ]
    [abl-startup-options]
```

`-folder` *folder-name*

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

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

`-db dbname [` *db-options* `]`

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

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

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

The `stsurlutil status` utility writes the status of the gateway URL to the file, `dbauthgateway_status.txt`. One line per database is written to the file in the format:

```
DbName,Enabled
```

stsurlutil update utility

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

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

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

`-url` *`url`*

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

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

`-servername` *`virtual-hostname`*

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

`-folder` *`folder-name`*

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

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

`-ssl` *`args`*

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

`-test true | false`

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

`-db dbname [` *`db-options`* `]`

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

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

Specify additional databases and connection options.

abl-startup-options

Specify additional ABL connection parameters.

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

Strong Password Encryption Utility (stspwdutil)

The `stspwdutil` utility generates strong (AES-128) encrypted passwords for inclusion into configuration storage and verifies an encrypted password against a clear text string.

```
stspwdutil
      { encrypt text-input [ key-pad-id ]
          |
            verify encrypted-string reference-string }
```

text-input

Specify a clear-text value to encrypt and return as a hex-binary encoded value.

Use operating system shell command line escapes and quoting to enter meta-character text.

key-pad-id

Specify an alternate Key Pad ID. The default Key Pad ID is zero (0), and valid values are the range of 0 - 9.

encrypted-string

Specify the hex-binary encoded value to verify.

reference-string

Specify a clear-text value to use in the validation process of *encrypted-string*. If you do not supply a *reference-string* value, `stspwdutil` returns an error.

When `stspwdutil` is executed with the `encrypt` option, it takes as input a clear text string and an optional Key Pad ID. The output is an encrypted string.

When `stspwdutil` is executed with the `verify` option, it takes as input an encrypted string, and a clear text string to use as a base to verify the encrypted string. The output is `true` or `false`.

The `stspwdutil` utility generates strong (AES-128) encrypted passwords. The output uses an encryption prefix whose algorithm type `'oep'` and an encoding type of `'h'` (for hex-binary encoding), and a Key Pad ID of 0-9. The encoded hex-binary is in blocks of 16 characters, which reflects its AES encryption algorithm's block size. So the encoded value's size does not reflect the actual size of the encrypted text data.

The encryption prefix of encrypted text has the form `aaaed::`, described as follows:

`aaa` — Designates the encryption algorithm type code

`e` — Designates the text encoding used for the encrypted value, `h` is specified for hex-binary

`d` — Designates the Key Pad ID used to generate an encryption key

`::` — The delimiter between the prefix and the encoded data

Examples

Examples of `stspwdutil` with the `encrypt` option:

```
proenv> stspwdutil encrypt password
oeph0::76E5F6C162276768465F02E4D2D1DDCD


proenv> stspwdutil encrypt password 3
oeph3::3B295B1855077677CB2ED521DD27C60D
```

Examples of `stsclientutil` with the `verify` option:

```
proenv> stspwdutil verify eph0::76E5F6C162276768465F02E4D2D1DDCD password
true


proenv> stspwdutil verify eph0::76E5F6C162276768465F02E4D2D1DDCD mypasswd
false
```

5

ABL constructs that support the OpenEdge Authentication Gateway

The following table summarizes the ABL constructs that have behavior specific to the OpenEdge Authentication Gateway and its implementation of a Security Token Service (STS). For more information on each, see *OpenEdge Development: ABL Reference*.

Construct	Behavior
`SET-DB-CLIENT` function	When using an STS-enabled database, `SET-DB-CLIENT` does not check the user identity against the ABL session domain registry. In this case, the identity is always checked against the database's own domain registry configuration.
`SET-USER-ID` function	When using an STS-enabled database, the `_oeusertable` authentication system is not required, and accounts do not have to be in the `_User` table.
`AUDIT-EVENT-CONTEXT` attribute of the Client-principal object handle	The value of `AUDIT-EVENT-CONTEXT` is stored in the `_Event-detail` field in the audit record.
`LOGIN-STATE` attribute of the Client-principal object handle	This attribute has five additional values to support STS: `NO-LOGIN`, `NO-ACCESS`, `REVOKED`, `DISABLED`, and `LOCKED`.
`SEAL( )` method of the Client-principal object handle	After an STS-enabled database is connected to a given session, you cannot call the `SEAL( )` method on the client principal with an encoded domain access code. After an STS-enabled database connection is made, you can only call `SEAL( )` with a clear text domain access code.

Construct	Behavior
`VALIDATE-SEAL( )` method of the Client-principal object handle	`VALIDATE-SEAL( )` generates an audit event when the method finds the client principal to be expired for the first time. When this happens, the method generates an audit event for a logout operation.
`Progress.Security.PAMStatus` class	To support STS authentication, five new properties have been added to this class: `AccessDenied`, `AccessRevoked`, `AccountDisabled`, `LoginDenied`, and `LoginLockout`.