Progress®



OpenEdge® Getting Started:
OpenEdge Authentication Gateway Guide

OpenEdge®

Copyright

© 2021 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.

DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Defrag This, Deliver More Than Expected, DevReach (and design), Icenium, Inspec, Ipswitch, iMacros, Kendo UI, Kinvey, MessageWay, MOVEit, NativeChat, NativeScript, OpenEdge, Powered by Chef, Powered by Progress, Progress, Progress Software Developers Network, SequeLink, Sitefinity (and Design), Sitefinity, Sitefinity (and design), SpeedScript, Stylus Studio, Stylized Design (Arrow/3D Box logo), Styleized Design (C Chef logo), Stylized Design of Samurai, TeamPulse, Telerik, Telerik (and design), Test Studio, WebSpeed, WhatsConfigured, WhatsConnected, WhatsUp, and WS_FTP 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, Chef Automate, Chef Compliance, Chef Desktop, Chef Habitat, Chef WorkStation, Corticon.js, Corticon Rules, Data Access, DataDirect Autonomous REST Connector, DataDirect Spy, DevCraft, Fiddler, Fiddler Everywhere, FiddlerCap, FiddlerCore, FiddlerScript, Hybrid Data Pipeline, iMail, JustAssembly, JustDecompile, JustMock, KendoReact, NativeScript Sidekick, OpenAccess, PASOE, Pro2, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Push Jobs, SafeSpaceVR, Sitefinity Cloud, Sitefinity CMS, Sitefinity Digital Experience Cloud, Sitefinity Feather, Sitefinity Insight, Sitefinity Thunder, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Supermarket, SupportLink, Unite UX, 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.

July 2021

Product version: Progress OpenEdge 12.4

Updated: 2021/06/30

Table of Contents

Preface....9

Learn about OpenEdge Authentication Gateway....11

Functional overview....13
Server Management....14

OpenEdge Authentication Gateway Tutorial....15

Preface to the OpenEdge Authentication Gateway tutorial....16
Install the OpenEdge Authentication Gateway server....18
Start the OpenEdge Authentication Gateway....20
Authenticate the test user....22
Enable the database to use the OpenEdge Authentication Gateway....24
Use a server key with the OpenEdge Authentication Gateway....30
Use a named domain for authentication....39
Use connection roles for authorization....55
Use encrypted passwords in the sts.properties file....59
Add auditing to the OpenEdge Authentication Gateway server....62
Add policies to the OpenEdge Authentication Gateway server....65
Replace the default key and certificate on OpenEdge Authentication Gateway server....70

Configure the OpenEdge Authentication Gateway....77

Configuration overview....78
Configure domains....79
Configure domain access codes....81
Domain configuration requirements for Native Token Exchange (SSO)....82
Configure LDAP Connections in a Security Token Service (STS)....83
Configure policies....85
Configure multi-domain user ID support....87
Configure event callbacks....88
Spring Security Configuration....91
Create and deploy a certificate for HTTPS with the OpenEdge Authentication Gateway Server....92
STS server key configuration....93
About STS client key management....97
Configure STS client key management using stskeyutil....98
Configure STS client key management using the AdminServer STSKey plugin....99
STS client key management logging....101

Configure the OpenEdge Database to use the OpenEdge Authentication Gateway....103

Configure your OpenEdge database....103
SSO Token Exchange....105
STS Client Key (for database server)....106
Start OpenEdge database server 107

Configure applications to use the OpenEdge Authentication Gateway.109

The STS AuthenticationProvider....109
Configuration and Testing....111
OpenEdge Authentication Gateway configuration....111
Test the connection....112
Configure a web application....113
Troubleshoot the PAS for OpenEdge instance....114

Reference entries....117

OpenEdge Authentication Gateway utilities....118
Change the keystore password utility (changeP12pwd)....118
Data export utility....119
STS client utility (stsclientutil)....120
STS Key Utility (stskeyutil)....123
STS Connection Role Utility (stsconnroleutil)....130
STS URL Utility (stsurlutil)....137
Strong Password Encryption Utility (stspwdutil)....140
ABL reference....141
SET-DB-CLIENT function (enhanced for STS)....141
SETUSERID function (enhanced for STS)....146
AUDIT-EVENT-CONTEXT attribute (enhanced for STS)....148
LOGIN-STATE attribute (enhanced for STS)....149
SEAL( ) method (enhanced for STS)....151
VALIDATE-SEAL( ) method (enhanced for STS)....153
Progress.Security.PAMStatus class (enhanced for STS)....155
AccessDenied property....156
AccessRevoked property....156
AccountDisabled property....156
AccountExpired property....156
AuthenticationAccess property....157
AuthenticationFailed property....157
Custom property....157
InvalidConfiguration property....157
LoginDenied property....157

LoginLockout property....158
MaxTries property....158
MissingCredentials property....158
PasswordExpired property....158
PermissionDenied property....158
Success property....159
UnknownUser property....159
Startup parameters....159
Certificate store path (-certstorepath)....159
Domain (-domain) (for STS capability)....160
Keystore path (-keystorepath)....160
No Host Verify (-nohostverify) (for STS capability)....161
No Session Reuse (-nosessionreuse) (for STS capability)....161
OS User (-OSUser) (for STS capability)....162
Security cache size (-secsize)....163
STS logging level (-stslogginglevel)....164
Database utilities....165
PROUTIL ENABLEAUTHGATEWAY qualifier....165
PROUTIL DISABLEAUTHGATEWAY qualifier....166
Auditing events for OpenEdge Authentication Gateway....166
Glossary of Terms....167
STSKey plugin for the AdminServer....170

Preface

Purpose

The *OpenEdge Authentication Gateway* guide describes <placeholder>

Audience

This guide is aimed at <placeholder>.

Organization

This book is organized into the following chapters::

- **Learn about OpenEdge Authentication Gateway**—
- **Configure the OpenEdge Authentication Gateway**—
- **Configure the OpenEdge Database to use the OpenEdge Authentication Gateway**—
- **Configure applications to use the OpenEdge Authentication Gateway**—

Documentation conventions

See Documentation Conventions for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content library.

1

Learn about OpenEdge Authentication Gateway

The OpenEdge Authentication Gateway bridges the gap between OpenEdge and 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 OpenEdge in order to produce native OpenEdge security tokens, ABL Client-Principals. The native OpenEdge security tokens are used in the connection authorization of the database server 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 OpenEdge releases.

Note: The optional OpenEdge Authentication Gateway is a licensed product that can be separately purchased from Progress. OpenEdge Authentication Gateway is included with the Progress Application Server for OpenEdge license.

Enhanced OpenEdge Database Connection Security

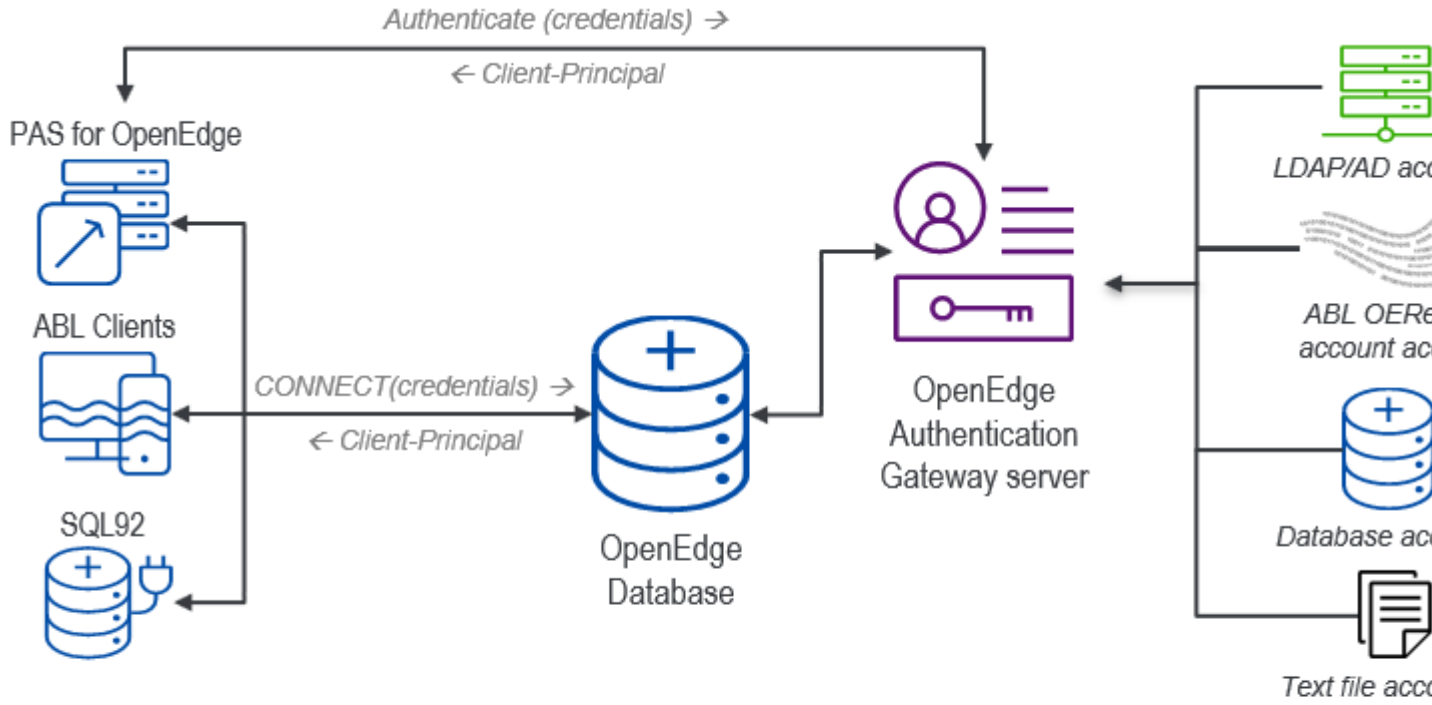
The OpenEdge Database includes support for enhanced client connection security through a network of interdependent OpenEdge product security features. This enhanced database connection security puts the Database Administration (DBA) in control of which OpenEdge clients may establish and use those connections in order to access the data on the server. 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 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 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 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 clients and the use of a network authentication service that may be shared with all OpenEdge databases and its clients.



Each client connection to the OpenEdge Database is authenticated by the OpenEdge Authentication Ga which can read user information from a user account source or directory information service such as Active Directory, OERealm, database, or text file accounts. PAS for OpenEdge can delegate user-ID/pas authentication to the OpenEdge Authentication Gateway.

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. An 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 and 12 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 PAS for OpenEdge servers.

The enhanced OpenEdge database connection security also increases the amount of database connection-related information available in its audit trail. 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 has been added.

For more information on the terms used in this guide, see the Glossary of Terms on page 167.

For details, see the following topics:

- Functional overview
- Server Management

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 distributed components of an ABL application. 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 an enterprise network.

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

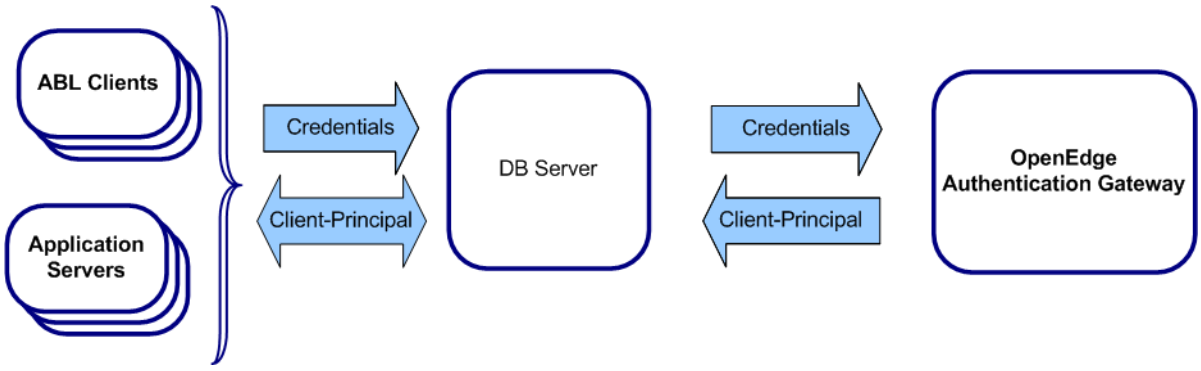
- Utilities to configure OpenEdge databases and applications to access OpenEdge Authentication Gateway services
- Tools to maintain and monitor OpenEdge database and application activity when using the Authentication Gateway
- ABL functions, handle attributes and methods, and class properties that support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

Note: On Linux platforms, you cannot start the OpenEdge Authentication Gateway from the root account. You must use an account that belongs to the group that has execute permissions in the `$DLC/bin` directory.

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

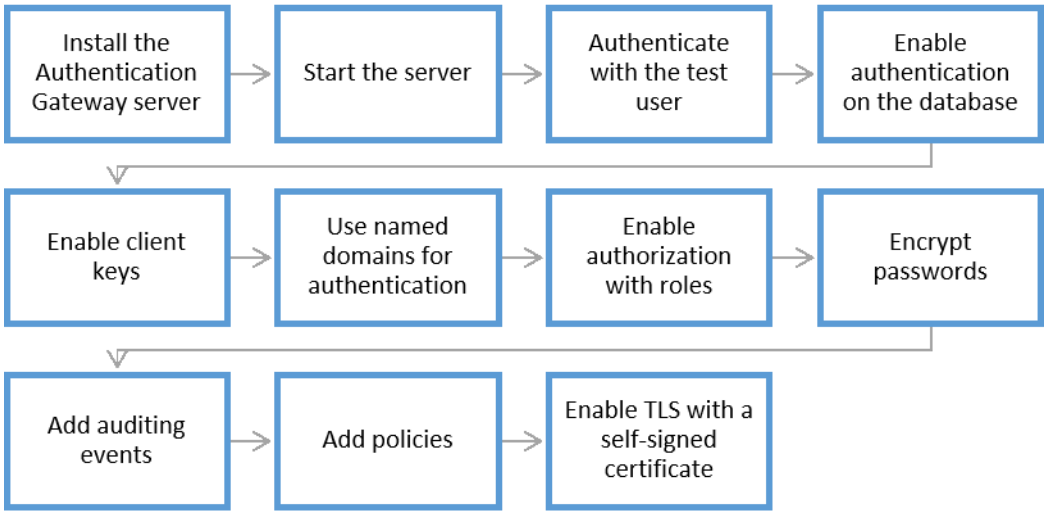
2

OpenEdge Authentication Gateway Tutorial

The goal of this tutorial is to enable you to get the OpenEdge Authentication Gateway server running, and to set up an OpenEdge database to be secured by the Authentication Gateway server. The OpenEdge Authentication Gateway provides enhanced database, application, and server security, which gives system administrators greater control over client connections to their resources.

For the purposes of this tutorial, you should have a separate Authentication Gateway server and OpenEdge database machines. The OpenEdge database accesses the Authentication Gateway server to authenticate access to its resources. For the purposes of this tutorial, you must also have the administrator-level credentials to log into the Authentication Gateway machine.

This tutorial covers these high-level tasks of getting started with the OpenEdge Authentication Gateway:



For details, see the following topics:

- Preface to the OpenEdge Authentication Gateway tutorial
- Install the OpenEdge Authentication Gateway server
- Start the OpenEdge Authentication Gateway
- Authenticate the test user
- Enable the database to use the OpenEdge Authentication Gateway
- Use a server key with the OpenEdge Authentication Gateway
- Use a named domain for authentication
- Use connection roles for authorization
- Use encrypted passwords in the sts.properties file
- Add auditing to the OpenEdge Authentication Gateway server
- Add policies to the OpenEdge Authentication Gateway server
- Replace the default key and certificate on OpenEdge Authentication Gateway server

Preface to the OpenEdge Authentication Gateway tutorial

Purpose

The *OpenEdge Authentication Gateway Tutorial* provides an introduction to the administrative activities that can be accomplished with the OpenEdge Authentication Gateway.

Audience

This tutorial is aimed at OpenEdge administrators, who have a license for the OpenEdge Authentication Gateway component of OpenEdge. The tutorial was written and tested with OpenEdge version 12.2, the Long Term Supported release of OpenEdge at the time of document preparation (August 2020), which will be supported for many years.

Prerequisites

In addition to the license for the OpenEdge Authentication Gateway, and some experience as an OpenEdge administrator, this tutorial assumes you are working in a distributed environment. You may want to provision a virtual machine with administrative privileges onto which you will install the OpenEdge with the Authentication Gateway component, and a separate machine to host the OpenEdge database, which will act as a client to the Authentication Gateway.

This tutorial uses the Windows operating system syntax and folder structure. Syntax may be different on your operating system for both the system-level commands and the proenv commands.

Documentation conventions

See Documentation Conventions for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content library.

Organization

This tutorial is organized into the following topics:

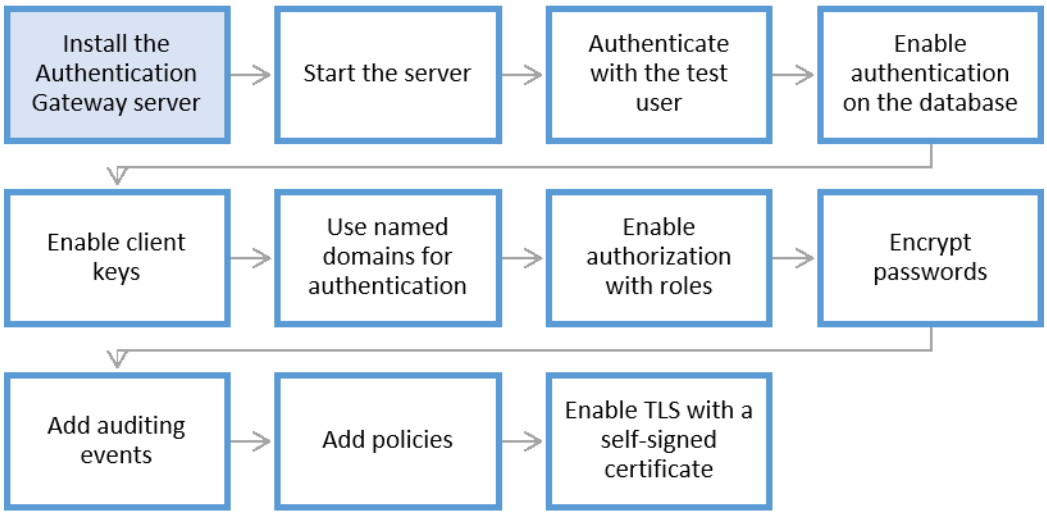
1. Install the OpenEdge Authentication Gateway
2. Start the OpenEdge Authentication Gateway
3. Authenticate with the test user
4. Enable an OpenEdge database to use the OpenEdge Authentication Gateway
5. Use a server key with the the OpenEdge Authentication Gateway
6. Use a named domain with the OpenEdge Authentication Gateway
7. Use roles for authorization in the OpenEdge Authentication Gateway
8. Use encrypted passwords in the sts.properties file
9. Add auditing to the OpenEdge Authentication Gateway server
10. Add policies to the OpenEdge Authentication Gateway server
11. Validate the OpenEdge Authentication Gateway server's Digital Certificate

Throughout the document, you will see numerous references to the OpenEdge database machine (or client database machine) and the OpenEdge Authentication Gateway server machine. For more information about this configuration:

- The database machine will host the sports2020 database. The tutorial suggests placing the sports2020 database in a folder called `/db` that you create in your WRK directory. On Windows, the default path would place this location at `C:\OpenEdge\WRK\db\`. All of the database commands should be run from this directory on the database machine. This also necessitates having OpenEdge installed on this client database machine. This machine is occasionally referred to as a "client machine", which is because the database acts as a client to the Authentication Gateway server's Security Token Service.
- The Authentication Gateway machine will host the Authentication Gateway server and Security Token Service. This machine must have OpenEdge installed with the Authentication Gateway component included. The default location for the Authentication Gateway server is in the WRK directory, and the default name of the server is `oeauthserver`. On Windows, the default path would place this location at `C:\OpenEdge\WRK\oeauthserver\`. This would be a separate machine from the database machine, so the `oeauthserver` and the `db` directories would not be in the same WRK directory.

Install the OpenEdge Authentication Gateway server

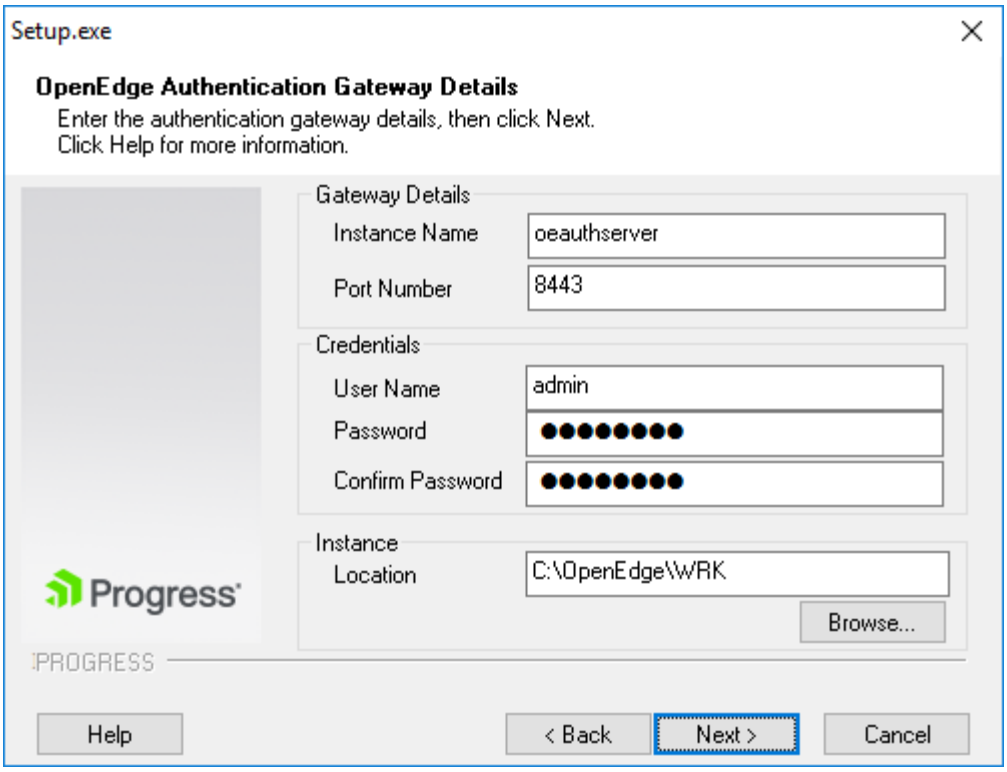
This topic describes the common installation settings for the OpenEdge Authentication Gateway server.



When installing the OpenEdge Authentication Gateway, which is configured as an individual component within the larger OpenEdge installation, take note of the instance name, port number, username, password, and location, since these will be used in the later topics in this tutorial.

To install the OpenEdge Authentication Gateway:

1. Ensure you have the proper 'licensing' in *Install OpenEdge* to access the OpenEdge Authentication Gateway as part of an OpenEdge installation. The Gateway component is known as `OE Auth Gateway` in the product licensing.
2. Install the OpenEdge product on the machine you want to use as the Authentication Gateway server. During the installation process, you are prompted to supply the following information on the OpenEdge Authentication Gateway Details page:



   Take note of the server defaults, or any changes you make to the defaults, as this information, specifically the Port Number and Instance Location, will be used throughout this tutorial.

   Because the Authenticate Gateway server is a type of PAS for OpenEdge instance, it is common to keep the `oeauthserver` instance in the `WRK` directory among other PAS for OpenEdge instances, such as the default PAS for OpenEdge instance, `oepas1`. For the purposes of this tutorial, we will keep the location in the `WRK`, which can be accessed using Proenv with the `%WRKDIR%` system variable on Windows, and `$WRKDIR` on Linux.

   Note: Port 8443 may already be in use on your system. If so, select an alternate port and copy that port number for use in the subsequent steps.

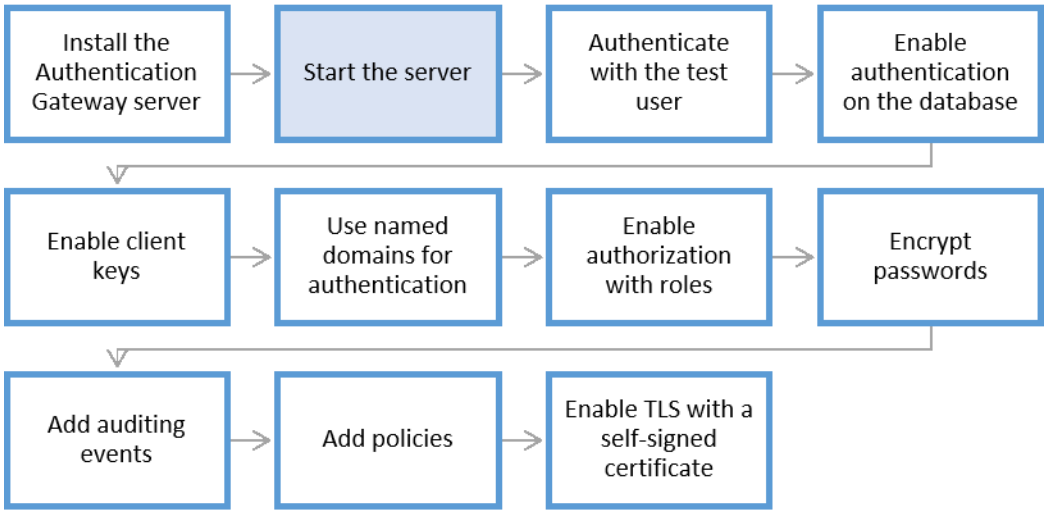
3. Continue with the OpenEdge installation as per your organization's requirements.

Summary

This topic introduced the installation process for the OpenEdge Authentication Gateway component. The Authentication Gateway is installed as part of an OpenEdge installation. Take note of the OpenEdge Authentication Gateway Details, as you will use them throughout this tutorial.

Start the OpenEdge Authentication Gateway

The OpenEdge Authentication Gateway is an instance of the Progress Application Server for OpenEdge that runs a Security Token Service. To start the OpenEdge Authentication Gateway server, you are just starting a registered PAS for OpenEdge instance.



To start the OpenEdge Authentication Gateway:

1. Open the Proenv command-line utility for the OpenEdge Authentication Gateway server machine that you completed in Install the OpenEdge Authentication Gateway server on page 18.
2. Change to Authentication Gateway instance directory. This path is provided during installation and is preconfigured with additional changes that support the Authentication Gateway server functionality. In a distributed environment, you could install the server where it is best for your system. In this case, navigate to the directory specified in the Authentication Gateway server installation, for example:

```
proenv>cd C:\OpenEdge\WRK\oeauthserver
```

3. Start the server using the `tcman pasoestart` command:

```
proenv>bin\tcman pasoestart
Starting stopped PASOE instance
............

Start action: start
Initial state: stopped
Initial processes: 0
Exit state: started
Exit description: Starting stopped PASOE instance
Exit processes: 33512 31300 4576
Exit status: 0
Exit errors:
```

Note: There is a limitation on Unix-like platforms that the OpenEdge Authentication Gateway cannot be started as the root user for security purposes.

As the the server starts, it loads the Security Token Service located in the `C:\OpenEdge\WRK\oeauthserver\webapps\ROOT` directory.

4. When the server is running, locate your machine name by using the `hostname` command:

```
proenv>hostname
yourgatewayhostname
```

Note: In some cases, you may need to access the Authentication Gateway hostname through the external IP address of the machine.

5. With the server running, use the client testing tool, `stsclientutil`, to ping the server:

```
proenv>stsclientutil -url https://yourgatewayhostname:port –nohostverify -cmd ping

ping ... OK
```

Note: The `stsclientutil` is best used on a separate machine from the Authentication Gateway server, such as the machine where you will create the OpenEdge database. The Authentication Gateway prohibits access using localhost or its IP address equivalent as the hostname.

Replace *yourgatewayhostname* in the above command with your hostname. The *port* is the port number that you specified during the installation. The default is 8443, but if you specified a different port during the installation, be sure to make that change.

The `-nohostverify` option allows access to the Authenticate Gateway using the server's default HTTPS certificate. This is for development and testing use only. Later in the tutorial, in the Replace the default key and certificate on OpenEdge Authentication Gateway server on page 70 topic, you replace this default certificate with a self-signed certificate. In production, you should use a digital certificate that is signed by a Certificate Authority.

The `ping … OK` message indicates that the server is up and running.

Summary

In this topic, you started the Authentication Gateway server instance that you had previously installed. By pinging the instance, using the `stsclientutil` utility (located in the OpenEdge installation's `/bin` directory) with the `-cmd ping` option, you verified that the server was started and accepting requests. This was a stand-alone test and had no interaction with any user account authentication system.

For now, you are using the `-nohostverify` option to suppress the error message raised when valid certificates are not available. Production installations require certificates to support secured communication between the Gateway and clients. You will add certificates in a later topic so that you can run the Gateway as designed.

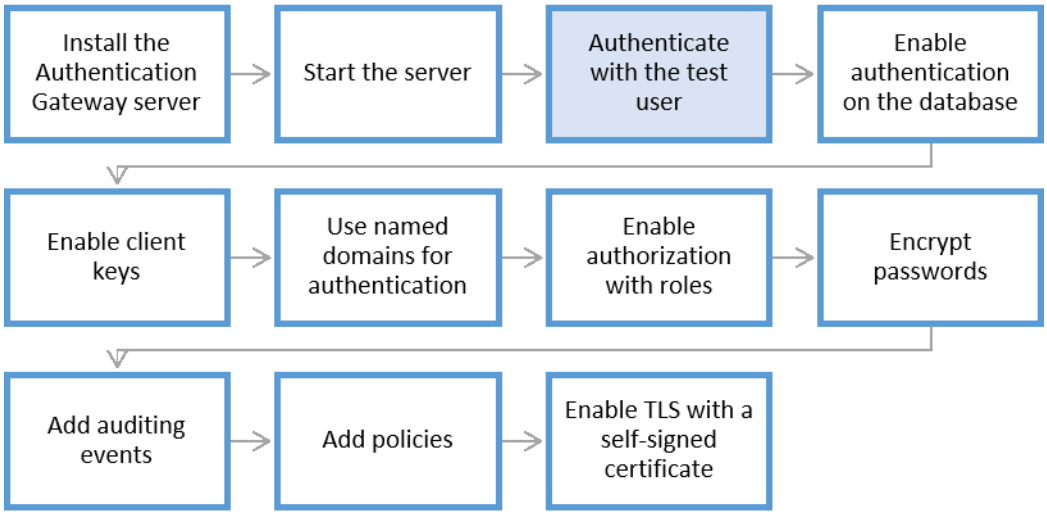
Using the `-help` option shows other options available for the command line utility:

```
proenv>%DLC%\bin\stsclientutil -help
stsclientutil v1.0.4
usage: stsclientutil [options]
  where [options] (* required)
 * -cmd <cmd> = [ping | authenticate | exchange]
 * -url <baseURL>
   -user <user[@domain]>
   -domain <domain>
   -sslversion <SSL version> = [TLSv1 | SSLv2 | SSLv3 | TLSv1.0 | TLSv1.1 | TLSv1.2 |
TLSv1.3]
   -sslciphers <SSL Ciphers>
   -installpath <oeInstallPath>
   -keystorepath <keyStorePath>
   -certstorepath <certStorePath>
   -clientkeyhdr <clientKeyHdrName>
   -logginglevel <loggingLevel>
   -nohostverify
   -servername <Host name which supports SNI and the same should configure on server>
   -help
```

In the next topic, we will use the `-cmd authenticate` option to authenticate a user based on the `users.properties` file.

Authenticate the test user

This topic covers testing the default OpenEdge Authentication Gateway configuration with the default user called `test` and a blank domain defined in the `users.properties` file. This configuration is available by default for users to explore the `stsclientutil` and the Authentication Gateway before locking down the database.



For this topic, the authentication details from the client can be completed on the same machine as the Authentication Gateway server machine. But in later topics, separate machines are required, and we will move away from the `users.properties` file to the operating system logins.

Authenticate the `test` user and password with `stsclientutil`

By default, we have configured the Authentication Gateway to authenticate with a blank domain and a `test` user defined in the `oeauthserver\webapps\ROOT\WEB-INF\config\users.properties` file. In this topic, we'll run a quick test with `stsclientutil` and test the settings defined in `domains.json` for the blank domain.

To authenticate the `test` user and password with `stsclientutil`:

1. Ensure your Authentication Gateway server is running on the server host machine.
2. In a Proenv console, use an Authentication Gateway client to pass a set of credentials to the Security Token Service for authentication using `stsclientutil`, for example:

```
proenv>stsclientutil -url https://yourgatewayhostname:port -nohostverify -cmd
authenticate -user test -password test
```

Note: Using the `stsclientutil` provides the best results when used from a separate machine from the Authentication Gateway server, such as the machine where you will host your OpenEdge database.

Replace *`yourgatewayhostname`* in the above command with the hostname of your Gateway server. The *`port`* is the port number that you specified for the Gateway server during the installation. The default is 8443, but if you specified a different port during the installation, be sure to make that change.

The `Authentication Succeeded` message indicates that the server is up and running.

The authentication succeeds because there is a pre-defined test user in the `%WRKDIR%/oeauthserver/webapps/ROOT/WEB-INF/config/users.properties` file. If you view this file, you can also test authentication which the other users defined in that file.

Attempt to authenticate as an `intruder` user and password with `stsclientutil`

It is best to test authentication with a negative case. In this step, we will attempt the same command with a user who is not a valid user. You can use any user name and password to test this. We will use intruder to demonstrate an outsider's attempt to be authenticated.

1. Ensure your Authentication Gateway server is still running on the server host machine.
2. In a Proenv console, use an Authentication Gateway client to pass a set of credentials to the Security Token Service for authentication using `stsclientutil`, for example:

```
proenv>stsclientutil -url https://yourgatewayhostname:port -cmd authenticate -user
 intruder -password intruder -nohostverify
```

   Replace *yourgatewayhostname* in the above command with the hostname of your Gateway server. The *port* is the port number that you specified for the Gateway server during the installation. The default is 8443, but if you specified a different port during the installation, be sure to make that change.

3. The following message prints to the console, which states that you were not authenticated because of bad credentials:

```
error: stsclientutil: authenticate error: error= 5 (http error)  detail= 400  ()
error: stsclientutil: sts service error:  error= (error_code:2)
error: stsclientutil: sts service error:  error_description= (Bad credentials)
Authentication failed. Reason = User authentication failed
```

Summary

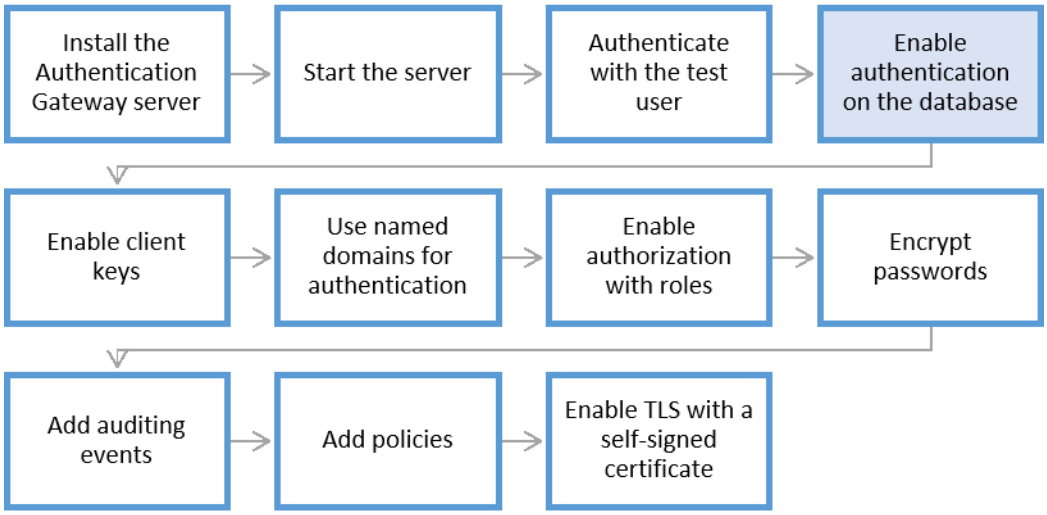
In this topic, you authenticated a default user with the name `test` and password `test` with a successful outcome. Additionally, you tested `intruder` as a username and password and the user authentication failed.

This demonstrates the basic format of the `stsclientutil` command, which uses the `-cmd` action with the `authenticate` option as well as `-user` and `-password` options to perform authentication based on the `users.properties` file located in the `%WRKDIR%/oeauthserver/webapps/ROOT/WEB-INF/config` directory.

This shows the authentication provided by the Authentication Gateway before enabling the feature in your database.

Enable the database to use the OpenEdge Authentication Gateway

This topic describes enabling the OpenEdge database to use the Authentication Gateway to secure client connections.



Enabling a database to use the Authentication Gateway requires several steps:

1. Create a copy of the sports2020 database.
2. Add the Authentication Gateway URL to the database.
3. Enable the database to use the Authentication Gateway.
4. Test a client connection.

Note: You can use the Progress Developer Studio for OpenEdge to access a GUI client and Database tools. Since much of the configuration of the Authentication Gateway is done on the command line, it was faster to test the connectivity with the `mpro` utility and accessing the character based Data Dictionary. Feel free to access the ABL client or Database activity using the graphical tools if you prefer.

Create a copy of the sports2020 database

For the purposes of this tutorial, we will create the sample sports2020 database in the working directory of our OpenEdge instance. This should be done on a separate machine from your Authentication Gateway server, so there is a separation between the Authentication Gateway server and the OpenEdge database machine.

1. On the database machine, use Proenv to navigate to the OpenEdge WRK directory, for example:

```
proenv>cd %WRKDIR%
proenv>cd
C:\OpenEdge\WRK
```

2. Make a new directory to separate the database, for example:

```
proenv>mkdir db
```

3. Navigate into this new db directory:

```
proenv>cd db
```

4. Create a copy of the sports2020 database in this location:

```
proenv>prodb sports2020 sports2020
Procopy session begin for Administrator on CON:. (451)
Database copied from C:\Progress\OpenEdge\sports2020. (1365)
Procopy session end. (334)
```

Note: If you created this database as an administrator, you may need to remove the read-only restriction on the db directory. To remove this restriction on Windows:

1. Open the Windows File Explorer and navigate to the WRK directory.
2. Right-click the db directory and select Properties.
3. Uncheck the Read-only attribute.
4. Click Apply.
5. Select Apply changes to folder, subfolder, and files.
6. Click Ok.

Add the Authentication Gateway URL to the database

Note: Before making any security changes to your database environment, it is a standard practice for database administrators to make a backup of their database. Ensure you have a current and tested backup of your database before proceeding with the next steps.

To add the Authentication Gateway server URL to enable Gateway authentication on your database:

1. The Authentication Gateway server must be running to register the URL for the Authentication Gateway server with the database. On the Authentication Gateway machine, ensure the instance is running:

```
proenv>cd %WRKDIR%\oeauthserver
proenv>bin\tcman pasoestart -restart
```

   When the startup completes with a zero-errors exit code, you can switch over to the database machine to add the Authentication Gateway URL to the database.

2. On the database machine, ensure you are operating in the directory that hosts the sports2020 database, for example:

```
proenv>cd C:\OpenEdge\WRK\db
```

3. The database cannot be running when you register the URL for the Authentication Gateway server with the database. Ensure the database is not running with the `proshut` command, for example:

```
proenv>proshut sports2020 -by
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
There is no server for database sports2020. (1423)
```

4. Run the following command to register the URL for the Authentication Gateway server with the database. The Authentication Gateway server must be running for the URL to be verified by the utility. Additionally, the database server must be shut down to use this command with the `-1` option:

```
proenv>stsurlutil update -url https://yourgatewayhostname:8443 -ssl -nohostverify
 -db sports2020 -1
[2020-07-19T21:06:56.846-04:00] DBSTSURL INFO OPERATION: UPDATE
[2020-07-19T21:06:56.846-04:00] DBSTSURL INFO SESSION:PARAM:
URL:https://yourgatewayhostname:8443,SSL-OPTIONS:-nohostverify
[2020-07-19T21:06:56.847-04:00] DBSTSURL INFO URL: https://yourgatewayhostname:8443
[2020-07-19T21:06:56.847-04:00] DBSTSURL INFO TEST-URL? yes
[2020-07-19T21:06:56.847-04:00] DBSTSURL INFO SSL-OPTIONS: -nohostverify
[2020-07-19T21:06:56.847-04:00] DBSTSURL INFO SERVERNAME:
[2020-07-19T21:06:56.954-04:00] DBSTSURL INFO STS Url created for "sports2020"
[2020-07-19T21:06:56.954-04:00] DBSTSURL INFO OPERATION COMPLETE
```

   This command loads the URL of the Authentication Gateway Server's Security Token Service (STS) into the database with the STS URL Utility. This command requires connecting to the database. In this example, we set the value with a single user mode connection (`-1`). Until we add certificates, we continue to use the `-ssl -nohostverify` option.

Enable the database to use the Authentication Gateway

With the blank domain as the default domain installed by the Authentication Gateway and the URL added, you are now ready to enable the use of the Authentication Gateway to enforce database security.

1. Ensure that you are operating in your database server directory, for example:

```
proenv>cd C:\OpenEdge\WRK\db
```

2. Run the following command to enable the Authentication Gateway server to secure your database:

```
proenv>proutil sports2020 –C enableauthgateway
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
Warning: It is recommended that connection role security enforcement be configured
when using the OpenEdge Authentication Gateway. (18519)
Authentication Gateway has been enabled for database sports2020. (12479)
```

Note: The database server cannot be running when you enable the Authentication Gateway.

This command enables the Security Token Service for your database. All database connections must now go through the Security Token Service to authenticate credentials before accessing your database. Additionally, you can add connection role security to limit access through authorization roles. This will be addressed in later topics.

3. Start the database server using the `–nohostverify` option, since the database is still using the sample certificates:

```
proenv>proserve sports2020 –nohostverify
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
21:25:00 BROKER     This broker will terminate when session ends. (5405)
21:25:00 BROKER     The startup of this database requires 40Mb of shared memory.
Maximum segment size is 1024Mb.
21:25:00 BROKER  0: Multi-user session begin. (333)
21:25:00 BROKER  0: Before Image Log Initialization at block 0  offset 1519. (15321)
21:25:00 BROKER  0: Login by administrator on CON:. (452)
```

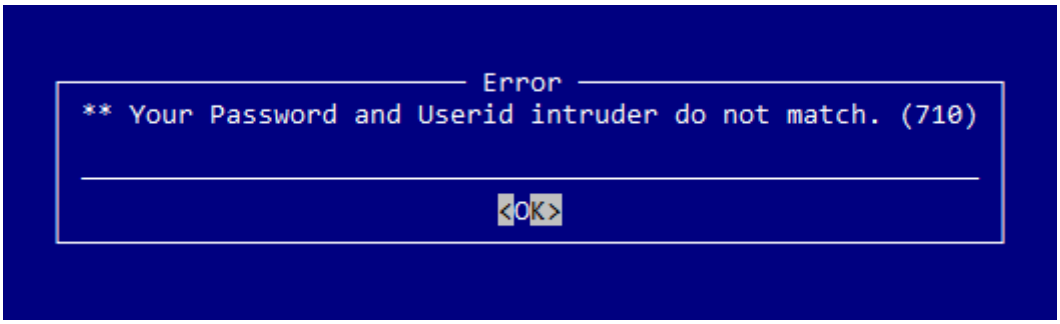
Test an ABL connection

The database is now ready to use the Authentication Gateway server's Security Token Service to authenticate, so let's test it using an ABL client.

1. With the database server running in multi-user mode, attempt to start a client session without supplying a correct user name and password:

```
proenv>mpro sports2020 -U intruder -P intruder
```

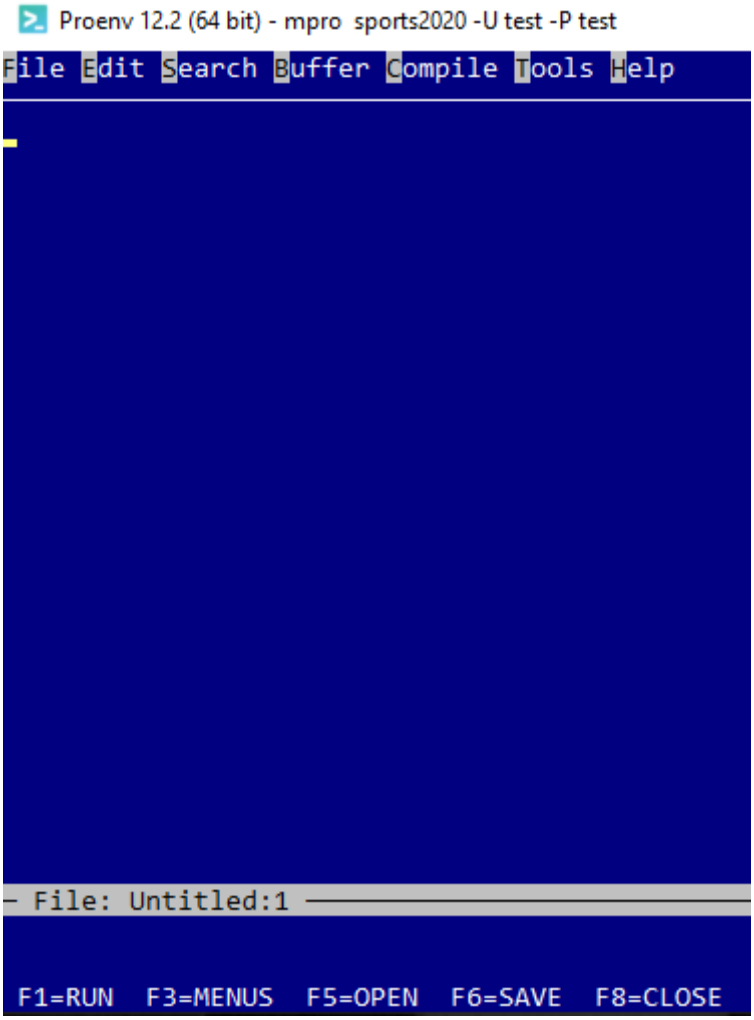
This produces the following error message:



2. Attempt to start a client session with a valid user name and password:

```
proenv>mpro sports2020 -U test -P test
```

Instead of an error message, you enter a procedure editor like the following:



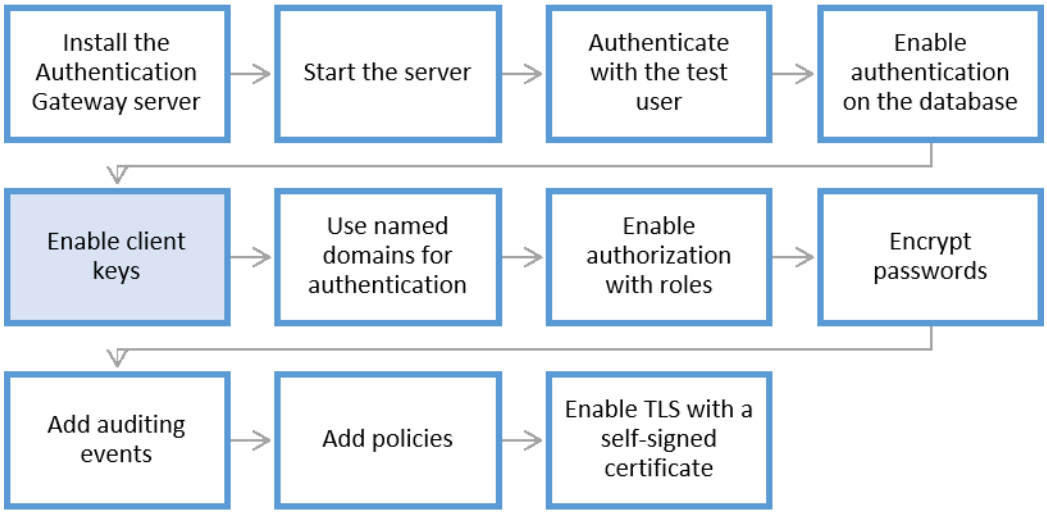
3. To exit the procedure editor, press **F3** to access the Menu, **F** to select File, and then **X** to exit.

Summary

In this topic, you created a database from the sample OpenEdge sports2020 database, added the URL for the Security Token Service, and effectively locked the database to outside access by forcing all connections to the database to go through the Security Token Service. This topic introduced the Authentication Gateway utility `stsurlutil` to specify that the database uses a specific URL for all connections to authenticate the database users. Then, we used the database utility `proutil -C enableauthgateway` to turn on the Gateway authentication. This topic also shows the importance of testing both the success and failure cases for connecting. The test user was again authenticated using the `users.properties` file on the Authentication Gateway server.

Use a server key with the OpenEdge Authentication Gateway

This topic describes how to create an Authentication Gateway server key, and then use the server key to generate a corresponding client key for the OpenEdge database machine.



Server keys are designed to prevent clients connecting to your Authentication Gateway server from an unauthorized machine. This practice is also known as “spoofing”. Server keys are yet another layer of protection provided by the OpenEdge Authentication Gateway server. Only clients with valid client keys, which are generated on the client machine using the Authentication Gateway server key, are able to connect to the Authentication Gateway server and access the Security Token Service (STS).

Note: After completing this topic, you will be required to have a client key (in the `%DLC%\keys` directory) on any client machine that authenticates user credentials using the Authentication Gateway server. If you **ever** need to disable the use of the server key and the `oests.server` domain:

- On the Authentication Gateway server machine, open the `sts.properties` file and set the `sts.server.key.enabled=false`.

Continue reading to learn more about these properties.

Complete the following steps to begin using server keys:

1. Create a server key on the Authentication Gateway server.
2. Configure the domain to use the key.
3. Install the client key on the database machine.
4. Test the client connections.

Create a server key on the Authentication Gateway Server

To create a server key on the Authentication Gateway server:

1. On your Authentication Gateway machine, use Proenv to navigate to the Authentication Gateway server instance directory, for example:

```
proenv>cd c:\OpenEdge\WRK\oeauthserver
```

2. Use `stskeyutil create` to create a server key file called `oests-key.ecp`:

```
proenv>stskeyutil create -file conf\oests-key.ecp
```

   This command prompts you for a password and domain access-code. Use the following credentials:

   - Server key file password: `password`
   - Server key's domain access-code: `passwordkey`

```
proenv>stskeyutil create -file conf\oests-key.ecp
Enter server key file password:
Enter server key's domain access-code:
Created server key file: conf\oests-key.ecp
```

Note: Keystrokes are not visible when typing for security. If you are worried you mistyped, you can delete the generated `oests-key.ecp` file and recreate it. You can use whatever combination password and access code you want. For the remainder of this tutorial, if you deviate from the recommended values, retain them for your records.

In a production environment, use a strong password and domain access-code.

   This command places the key file in the `oeauthserver\conf` directory, because that is the default location specified by the `sts.properties` file. In a production environment, the file would be placed in a protected directory where only Authentication Gateway server administrators could access it.

Enable the oests.server domain

The `domains.json` file is used to look up a particular domain and determine the configuration of that domain. This is a JSON file that the Authentication Gateway uses to look up valid domains for your instance.

Note: Enabling the `oests.server` domain will require your client machine to have a corresponding client key in its `%DLC%\keys` directory.

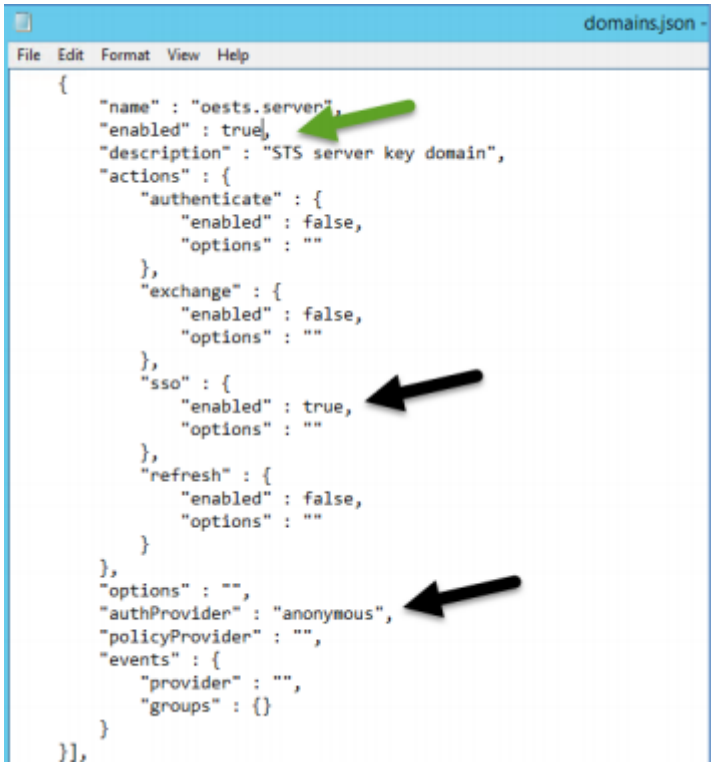
In following steps, you will enable a new domain by editing the existing `oests.server` entry:

1. Enable the `oests.server` domain by completing the following steps:

a. Make a copy of the original `C:\OpenEdge\WRK\oeauthserver\webapps\ROOT\WEBINF\config\domains.json` file, and name it `domainsORIG.json`, in case you want to revert to the original json at any point.

b. In a text editor, open the `C:\OpenEdge\WRK\oeauthserver\webapps\ROOT\WEBINF\config\domains.json` file. Open the file and search for `"name" : "oests.server"`.

c. Set the `oests.server` domain's "`enabled`" to `true`, as shown here:



Note: The other options required but already set by default are the sso `"enabled" : "true"` and `"authProvider" : "anonymous"`. Do not change the `authProvider`. It must be anonymous to work.

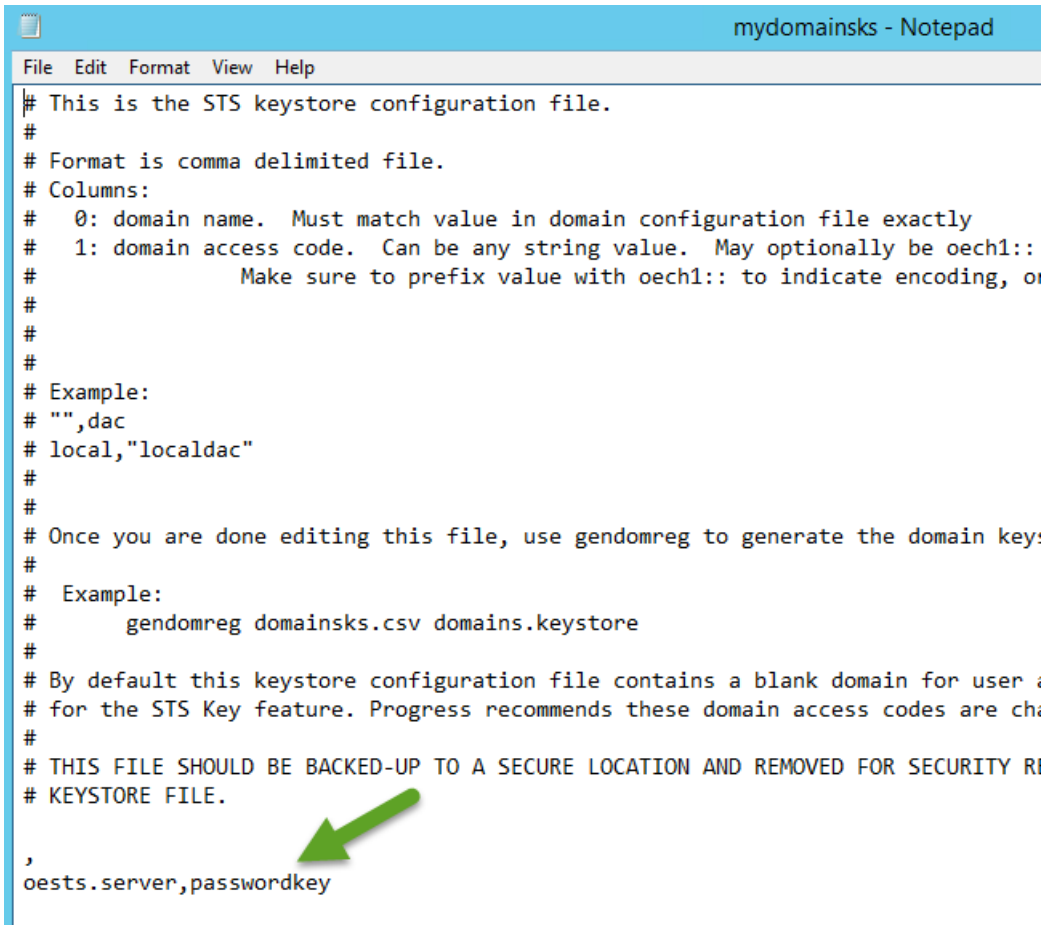
d. Save `domains.json`. We did not modify the name of this JSON file because it referenced in the `sts.properties` which is used by the Authentication Gateway to customize its properties.

Note: If you are worried that you may have made an edit to the JSON file that corrupted the file, you can use an online JSON editor to check that your JSON is valid.

2. Add the domain and password to the keystore configuration file:

a. On your Authentication Gateway server instance, open the `oeauthserver/webapps/ROOT/WEB-INF/config/domainsks.csv` file

b. Edit the file to replace `changeme!` with `passwordkey` that you just created as the server key access code for the domain, for example:



Note: The line in the `mydomainsks.csv` file with only a comma (`,`) is a blank domain with a blank domain access code. The blank domain is pre-defined in the domains.json file with the entry `"name" : ""` and `"authProvider" : "properties"`

c. Save the file as `mydomainsks.csv`. Using a custom name allows you to create `.csv` files specific to your environment.

3. Generate the keystore for the Authentication Gateway server:

a. In a Proenv command prompt, change to the `config` directory.

```
proenv>cd C:\OpenEdge\WRK\oeauthserver\webapps\ROOT\WEB-INF\config
```

b. Run the following command to create an encrypted keystore called `domains.keystore`:

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

```
proenv>gendomreg mydomainsks.csv domains.keystore
OEDomainRegistryUtil v1.5.3 (01/30/2019)

  [Generate domain registry from inputFile]

input file      : mydomainsks.csv    [CSV]
output file     : domains.keystore    [binary]
```

The output of the `gendomreg` utility is the `domains.keystore` file.

The `.csv` file is used to create an association between the `oests.server` domain in the `domains.json` and the domain access code. This domain access code is a secret that both the Authentication Gateway server and the client OpenEdge database know, which is used to verify client-principal authenticity.

The `.csv` file contains all of the OpenEdge Domain names and their domain access code that the Authentication Gateway server is permitted to work with. The `.csv` file is compiled into an encrypted `domains.keystore` file for security purposes. This encrypted keystore file is what would be deployed into an ITOps server. The `.csv` file with clear-text OpenEdge Domain access codes would not be deployed to an ITOps server, but retained for restricted access by ITOps administrators. When the Authentication Gateway server receives a client-principal of an individual who is asking for it to do something, the Gateway server looks in the `domains.keystore` file to find the OpenEdge Domain specified in the client-principal. If it is found, then the Gateway server uses the OpenEdge Domain's access code to validate the client-principal before it is accepted and used.

The `mydomainsks.csv` file should be moved somewhere safe and not kept with either the Authentication Gateway server or the database because it contains clear text password information.

4. Edit the `sts.properties` file to enforce the use of the keystore:

   a. Make a copy of the `C:\OpenEdge\WRK\oeauthserver\webapps\ROOT\WEB-INF\config\sts.properties` named `sts.ORIG.properties` in case you want to go back to the original settings.

   b. Open the `sts.properties` file in a text editor.

   c. Enable the use of server keys by setting `sts.server.key.enabled` to `true`:

```
# Optional: Enable client/server access key validation for clients of the
# STS application.  Set this to true to require the use of the server key
# file configuration and validation of STS client acess-keys.
#
# Set this property to true to authenticate individual OE installation's
# access to use the STS application and block fake/rouge clients.
#
# Defaults to false.
sts.server.key.enabled=true
```

   d. Specify where to find the key file by setting `sts.server.keyfile` to the key file path that was set earlier in the `oeauthserver/conf` directory, for example:

```
# Optional: STS server key file location. This propery is required if the
# sts.server.key.enabled property is set to 'true'.
#
# Default name is oests-key.ecp.  Specific full file-system path to file
# to ensure it can be found properly.
#
# Use the bin/stskeyutil utility to generate this encrypted Server access key
# file, which will be used to generate and install the STS applicaiton client's
# access-keys.  Must be specified if sts.server.key.enabled=true
#
```

```
# ( make sure to use forward slashes in file paths and URLs, even on windows )
sts.server.keyfile=C:/OpenEdge/WRK/oeauthserver/conf/oests-key.ecp
```

Note: You must use forward slashes even on Windows when specifying the path name.

e. Provide the Authentication Gateway server's key file password by setting the `sts.server.keyfilepassword` property to the password that was specified at server key creation, for example:

```
# sts key file password. Must be provided if server key file is specified in
# sts.server.key.enabled=true .
# May be be oech1:: encoded
sts.server.keyfilepassword=password
```

Note: In a production environment, use a stronger password to seal with key file. In a later topic, this tutorial explains how to Use encrypted passwords in the sts.properties file on page 59.

f. Save the file.

5. Restart the Authentication Gateway server. The `sts.properties` changes are not actively monitored for updates, so a restart of the server is required to pick up changes to this properties file.

a. Change to the Gateway server directory, for example:

```
proenv>cd C:\OpenEdge\WRK\oeauthserver
```

b. Run `tcman pasoestart -restart` to stop the server, clean the log files, and then restart the server:

```
proenv>bin\tcman pasoestart -restart
```

Using `tcman pasoestart -restart` shows the running processes in the `Exit processes` line. The multiple exit processes indicate the server and security token service are running, for example:

```
Stop running PASOE instance oeauthserver
.
Restarting the running PASOE instance oeauthserver
..............

Start action: restart
Initial state: started
Initial processes: 9008 5208 6244
Exit state: started
Exit description: Restarting the running PASOE instance oeauthserver
Exit processes: 8424 2564 6320
Exit status: 0
Exit errors:
```

Install the server key on the OpenEdge database machine

In a distributed system, you need to copy the Authentication Gateway server key (`oests-key.ecp`) to the OpenEdge Database machine that acts as a client to the Authentication Gateway server.

These steps demonstrate how to to copy and install the server key onto the OpenEdge database machine:

1. Copy the Authentication Gateway server key (`/oeauthserver/conf/oests-key.ecp`) from the server machine to the client machine that hosts the OpenEdge database. Copy the file into the client machine's `%DLC%\keys` directory and then verify the location of the `oests-key.ecp` file on the database machine, for example:

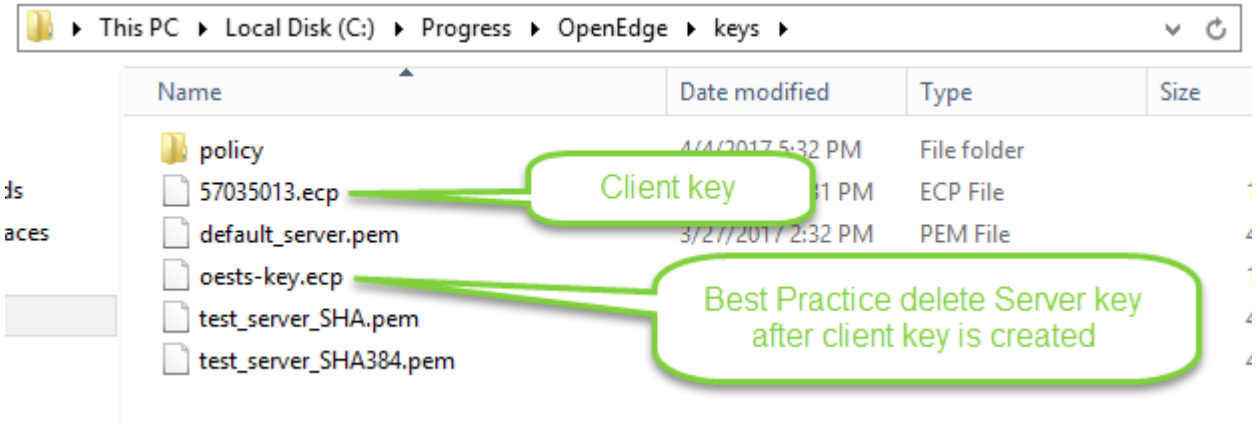
```
proenv>cd %DLC%\keys
proenv>dir *.ecp

 Directory of C:\Progress\OpenEdge\keys

07/19/2020  10:10 PM               224 oests-key.ecp
               1 File(s)            224 bytes
               0 Dir(s)  72,158,228,480 bytes free
```

2. On the client machine, use the `stskeyutil install` command to install the server key as a client key using `password` as the password (as was set at key file creation), for example:

```
proenv>stskeyutil install -file c:\Progress\OpenEdge\keys\oests-key.ecp -url
https://yourgatewayhostname:8443
Enter server key file password:
Created client key file
```

Note: A best practice is to immediately delete or move the `oests-key.ecp` file from the client machine to prevent others from accessing it.

The following screen-capture shows an example of the `%DLC%\keys` directory after generating the client key from the encrypted server key (`oests-key.ecp`):



Once you generate the client key, you should immediately delete the encrypted server key.

Install the key on the multiple distributed client machines

If you are running on a distributed system with more than two machines, repeat the "Install the server key on the client OpenEdge database machine" steps on each client machine where you want to secure your database with the Authentication Gateway. Then, test each client machine with the following procedure.

Test the client connections with a client key

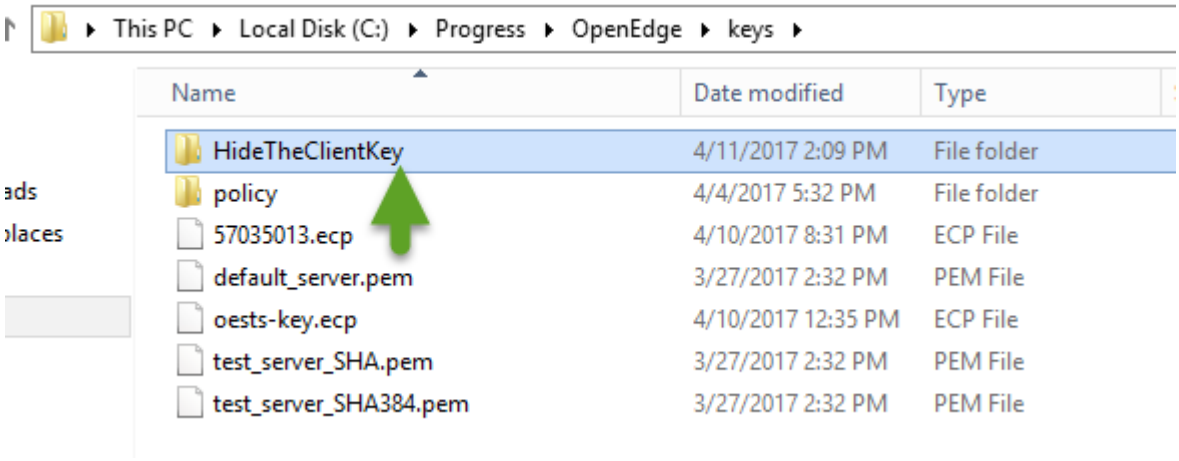
With a valid server key on the Authentication Gateway server, and a valid client key installed on the client OpenEdge machine (or machines), attempt to connect to the Authentication Gateway server using the client key.

1. From the client database machine, using the `stsclientutil` to ping the server to prove that the client key is being used by the Authentication Gateway:

```
proenv>stsclientutil -url https://yourgatewayhostname:8443 -nohostverify -cmd ping
ping ... OK
```

2. To test locking out a user without a valid server key:

   a. Temporarily move the client key to a subdirectory, for example, one called `HideTheClientKey`:



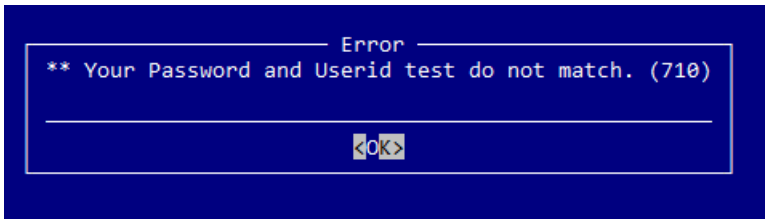
   b. Try to ping the server again using the `stsclientutil`:

```
proenv>stsclientutil -url https://yourgatewayhostname:8443 -nohostverify -cmd
ping
error: stsclientutil: ping error: error= 5 (http error)  detail= 401  ()
```

   The Authentication Gateway server rejects the ping as it is an unauthorized request.

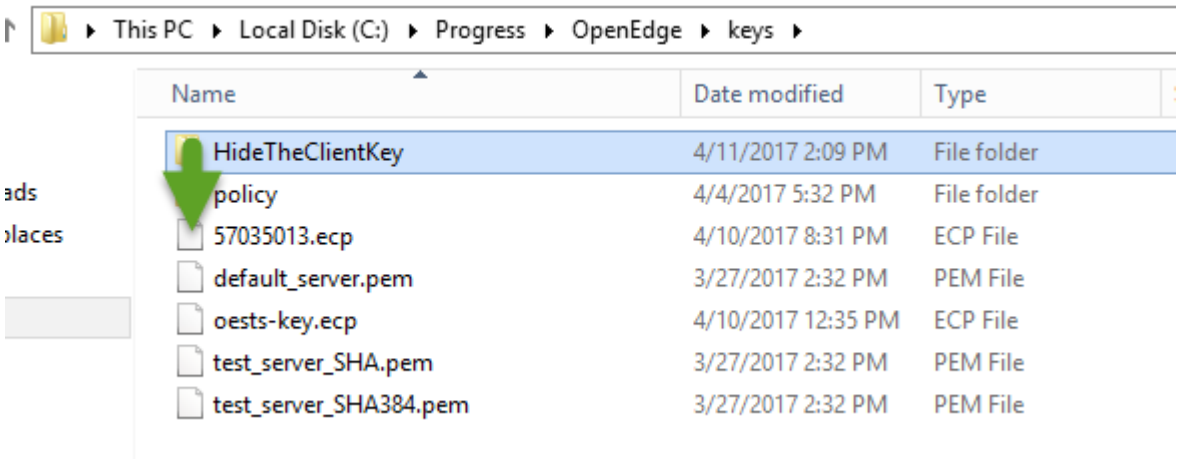
   c. Try to connect as the valid user, `test`:

```
proenv>mpro C:\OpenEdge\WRK\db\sports2020 –U test –P test
```



Note: The error message does not provide details about the missing server key. It leaves that investigation up to the database administrator to determine why the user was rejected.

3. Place the valid client key back in the `keys` directory.



4. Verify you can now connect using the valid user:

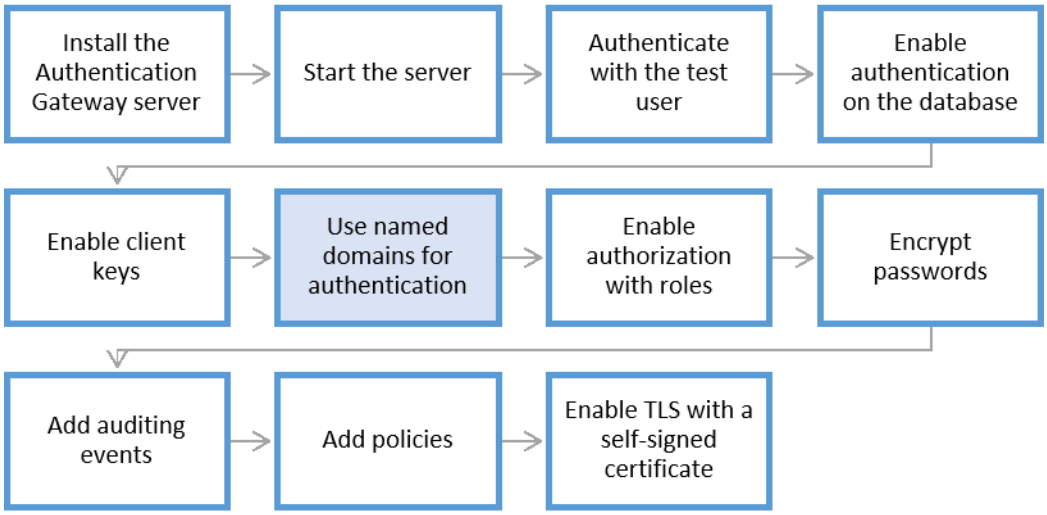
```
proenv>mpro C:\OpenEdge\WRK\db\sports2020 –U test –P test
```

Summary

In this topic, you created a server key using the `stskeyutil` utility. Using the `domains.json` file, you configured the `oests.server` domain to require a server key. You created a custom `mydomainsks.csv` file to create your server key and put it in the keystore using the`gendomreg` utility. Then, by modifying `sts.properties` file, you made that instance require the key and specified the key file location and password, and then restarted the server to reload the properties. You also installed the key using the `stskeyutil` tool, then tested access with the key. Optionally, we encouraged you to test removing the client key, noting that the message of a missing key is purposefully vague to be resolved by a database administrator. The client key must be kept in the `%DLC%\keys` directory of each client machine that uses the Authentication Gateway for authentication.

Use a named domain for authentication

This topic provides an introduction to using domains with the Authentication Gateway and database. This topic describes how to enable a domain that uses local operating system credentials to manage access to the Authentication Gateway server and database.



Domains exist in two places, the OpenEdge Authentication Gateway Server's Security Token Service configured with the `domains.json` file, and in the OpenEdge database. Up until now, you have been using the blank domain shipped in the `domains.json` file, and a pre-existing blank domain in the sports2020 database. In this topic, we add a domain called `local` to the `domains.json` file, and authenticate to the database using the operating system credentials of the Authentication Gateway server machine.

In this topic, we will complete the following steps:

1. Shut down and disable the use of the Authentication Gateway
2. Create a security safety net
3. Add a local domain
4. Add named security administrators
5. Add the local domain to the `domains.json` file
6. Restart the Authentication Gateway server
7. Enable the use of the Authentication Gateway on the database
8. Test database connection

Shut down and disable the use of the Authentication Gateway for the database machine

To separate any issues arising from running with the Authentication Gateway, take a moment to disable the Authentication Gateway on your database:

1. If your database is running on your client machine, shut it down:

```
proenv>cd %WRKDIR%\db
proenv>proshut sports2020 -by
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
Shutdown is executing. (1613)
Shutdown complete. (1614)
```

2. Disable the authentication gateway:

```
proenv>proutil sports2020 -C disableauthgateway
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
fmaFeatureDisable: The feature Authentication Gateway has been disabled. (11175)
Authentication Gateway has been disabled for database sports2020. (12490)
```

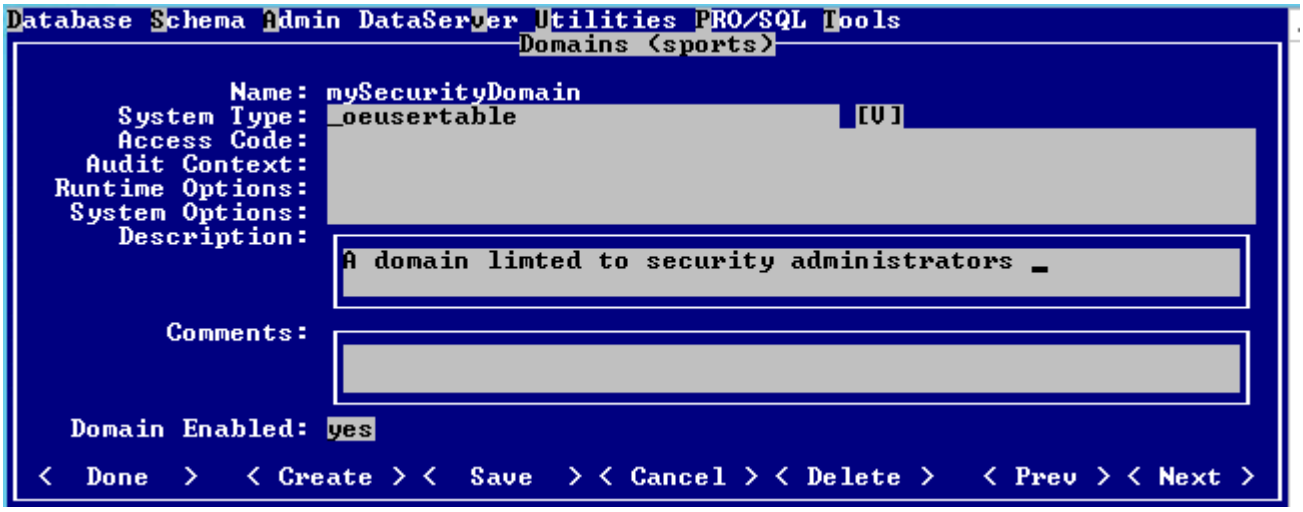
Create a security safety net

Because many security changes have the potential to lock you out of the database completely, you may want to define a security safety net domain, a security safety net administrator, and add that user as a named security administrator to protect you from inadvertently locking yourself out of the database.

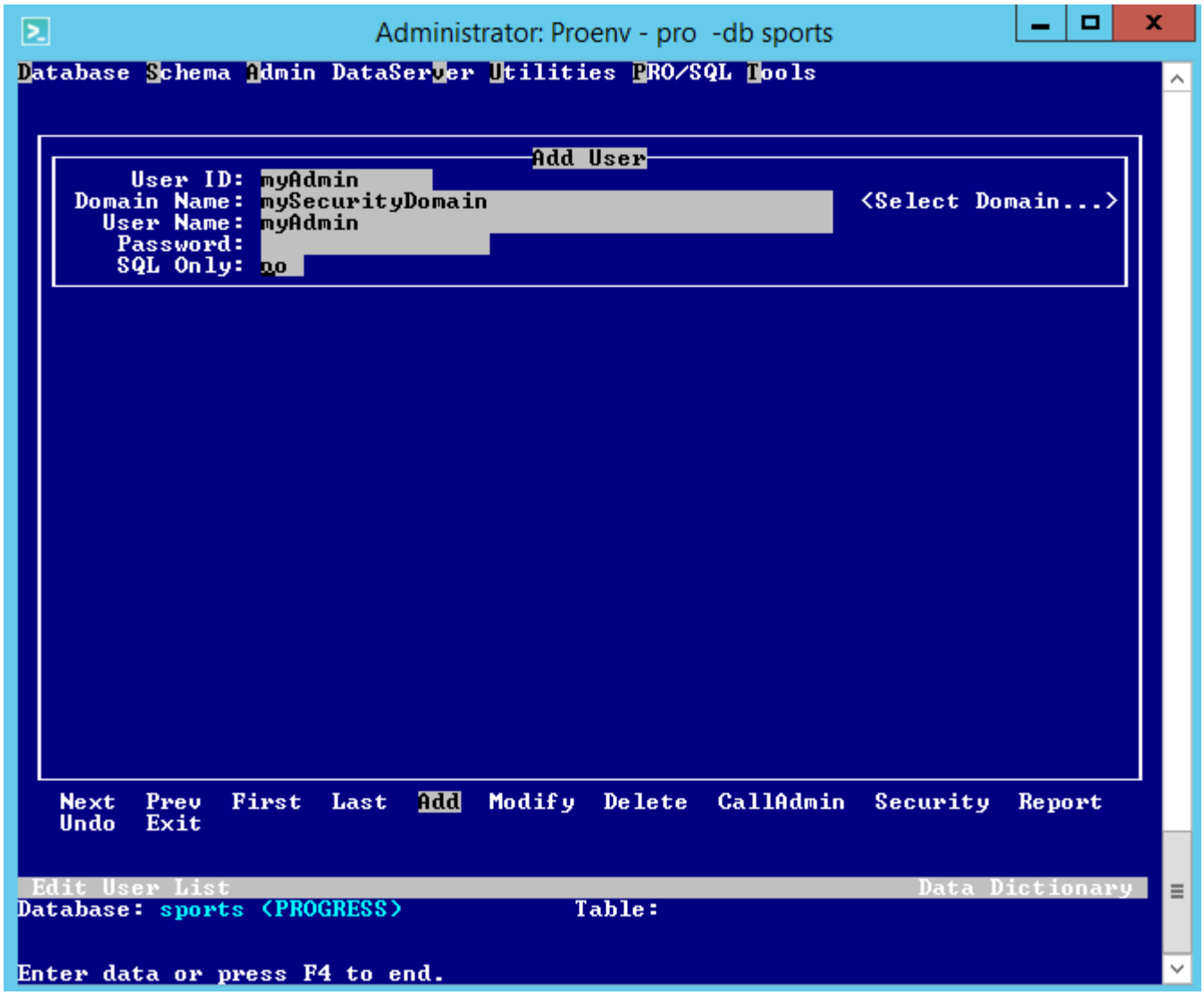
1. On your database machine, connect to the database in single user mode:

```
proenv>pro -db sports2020
```

2. Open the Data Dictionary by pressing **F3** > **T** (Tools) > **D** (Data Dictionary).
3. Enter the **Domains** menu by pressing **A** (Admin) > **S** (Security) > **M** (Domain Maintenance) > **D** (Domains).
4. Use the Tab key to highlight **< Create >** and then press Enter.
5. Provide the following information:
   a. Name: **mySecurityDomain**
   b. System table: **_oeusertable**
   c. Access code: **mySecurityDomain**
   d. Description: **A domain limited to security administrators**
   e. Use Tab and Enter to select **< Save >**.



6. Select **Done**.
7. Select **Done**.
8. Using the Data Dictionary, navigate to the **Edit User List** menu by pressing **A** (Admin) > **S** (Security) > **E** (Edit Users List).
9. Use the Tab key to select **Add**, and press Enter.
10. Add the following information:
    a. User ID: **myAdmin**
    b. Domain: **mySecurityDomain**
    c. Username: **myAdmin**
    d. Password: **myAdmin**

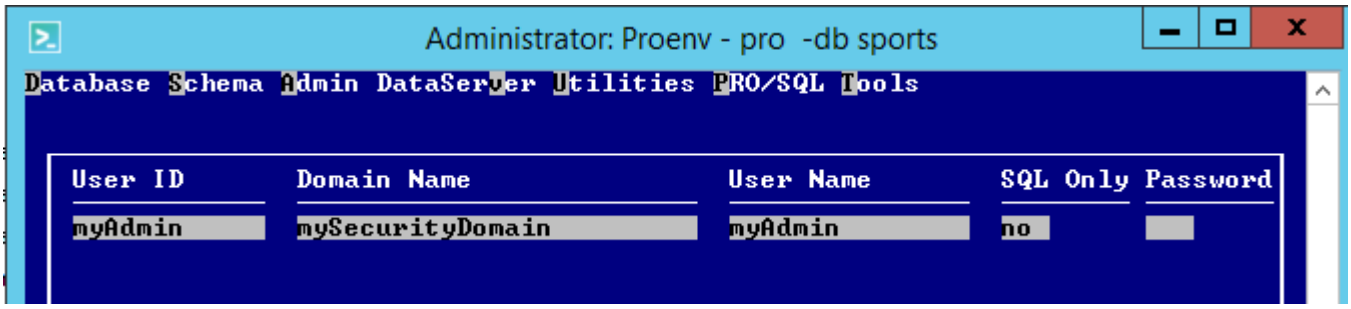


11. Press **F1** to save.

12. Verify the password.



13. Select **OK** to save.



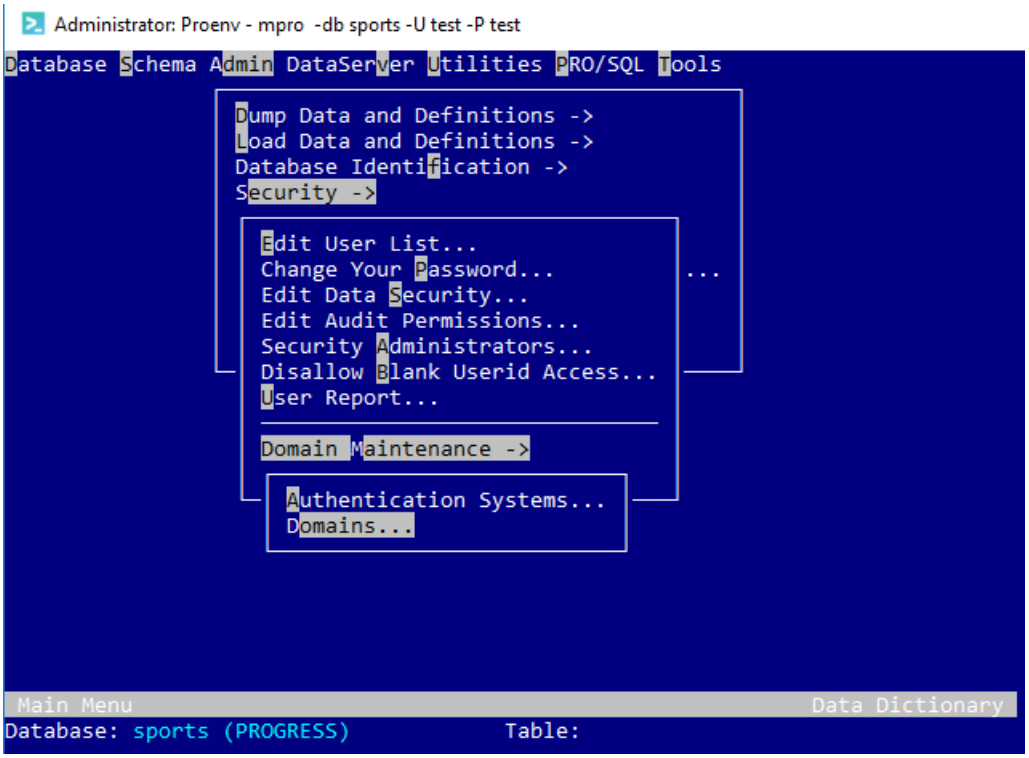
14. Press **E** to exit.

In a production environment, you would want to have multiple security administrators. For the purposes of the tutorial we have only added a single user.

Add a local domain to the database

In this task, you will connect to the sports2020 database using the Data Administration tool on your database machine. We will add a domain that allows you to use the operating system login credential to connect to the database.

1. In the Progress Database window, press **F3** > **A** (Admin) > **S** (Security) > **M** (Domain Maintenance) > **D** (Domains).



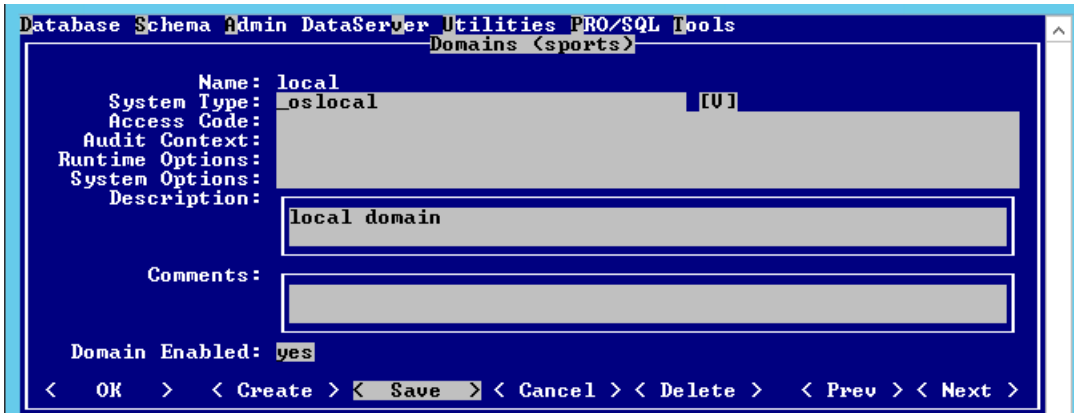
2. Tab to **< Create >** and press Enter.
3. Enter the following information
   a. Name: **local**
   b. System Type: **_oslocal**

Note: To access the drop-down menu that specifies the System Type options, tab to the System Type field and press the down arrow key.

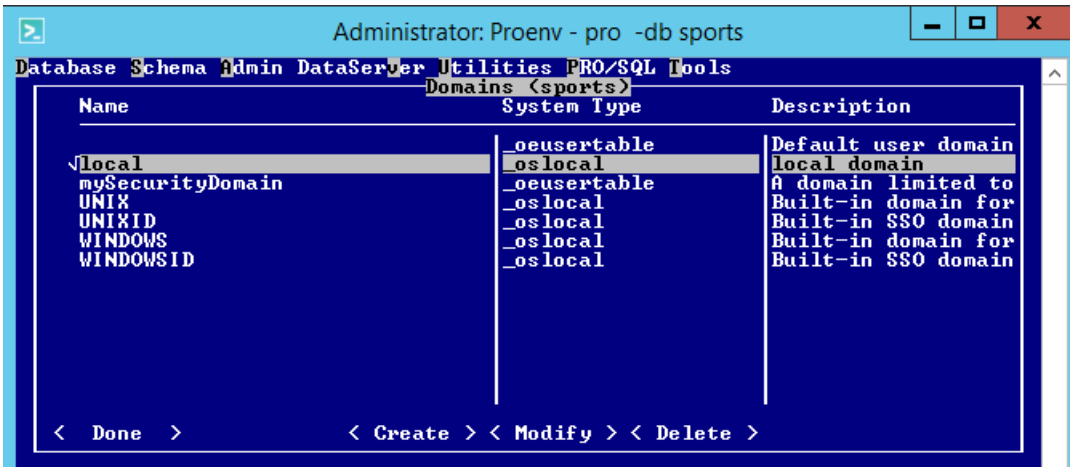
c. Access code: **localkey**

Note: The access code appears as blank. It must be the same as the access code you defined for the local domain in the `domainsks.csv` file.

d. Description: **local domain**

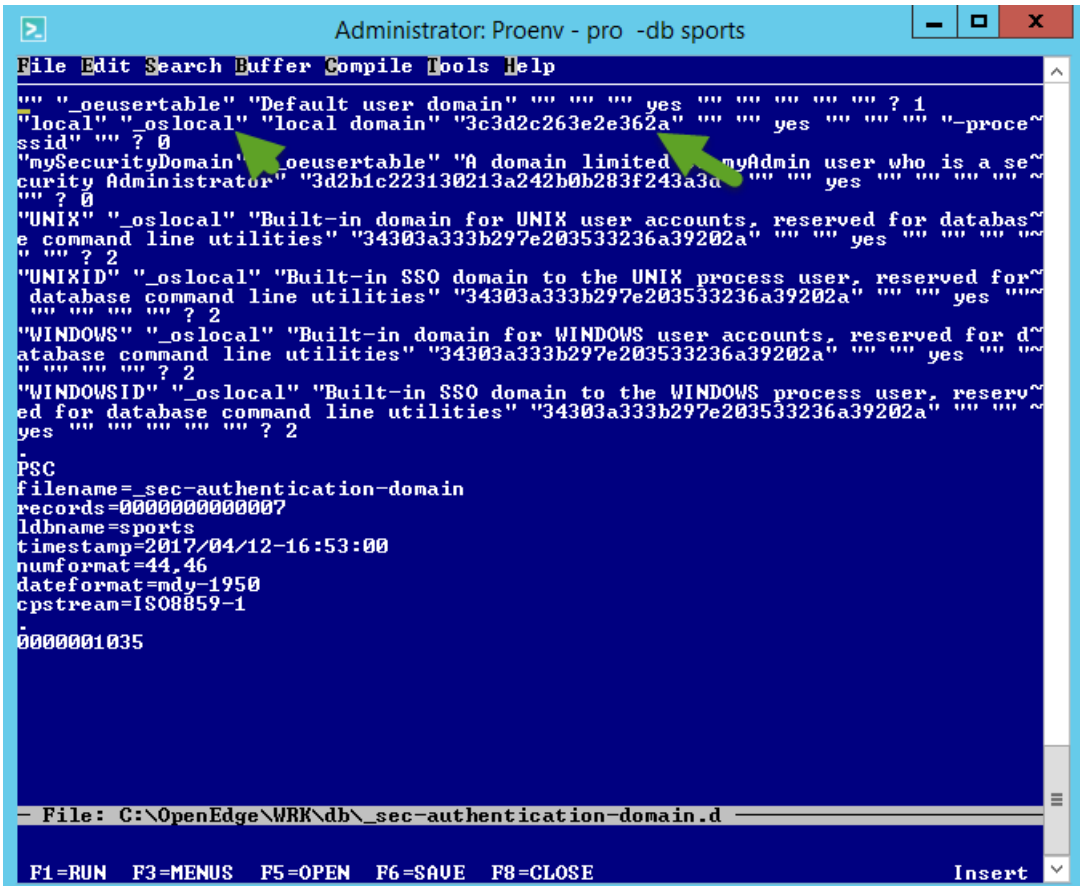


4. Select **< Save >** to save the changes.
5. Select **< Done >**.



6. Select **< Done >**.
7. (Optional) You can use the Data Administration Tool to dump the newly created records to a file on your local machine. Press **A** (Admin) > **D** (Dump Data and Definitions) > **Y** (Security Domains). If you specify no output directory, the files will be generated in the `C:\OpenEdge\WRK\db` directory that you launched the Progress window from.

   For your benefit, we have included a screen capture of the output. You can see the domain access code was encrypted. In this case, it is stored as combination of numbers and letters.



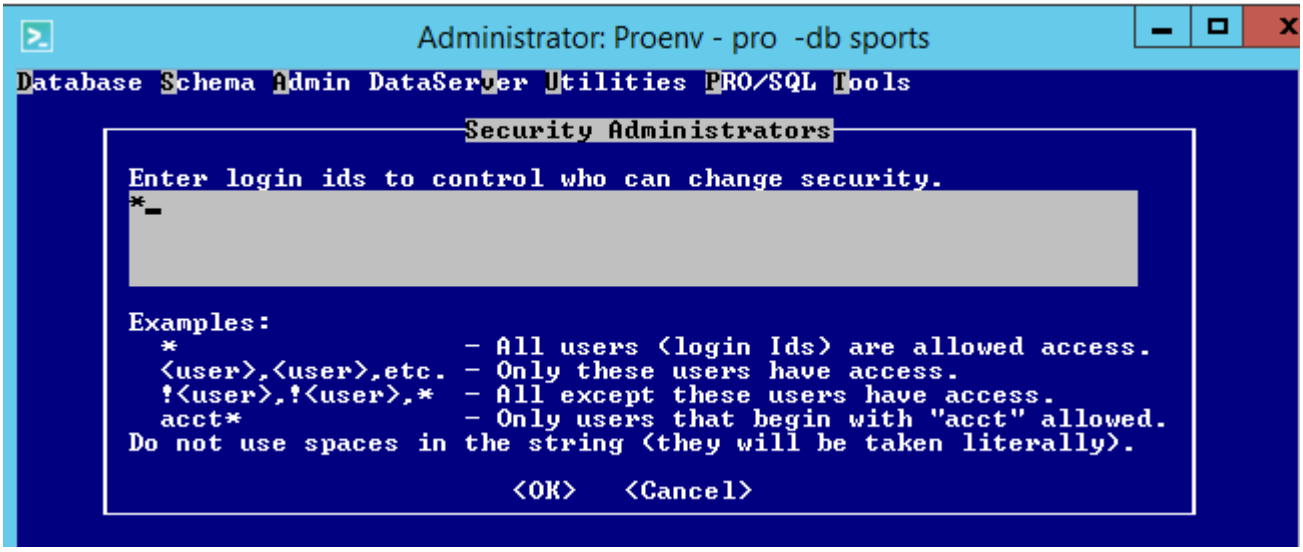
You are now ready to configure the OpenEdge Authentication Gateway server's STS to use the newly added domain.

Adding named security administrators inside the database

In these steps, you will create a list of users who will have permission to perform security administrator duties like add new Security administrators. We will add the `myAdmin` user with the domain `mySecurityDomain` to avoid locking ourselves out of the database.

Additionally, we'll add the `administrator` user on the Authentication Gateway server. We'll add two ways to refer to the user. The first, `administrator@local`, references the local domain. The second, `administrator`, using a blank domain.

1. On the database machine's Data Dictionary, select **A** (Admin) > **S** (Security) > **A** (Security Administrators) and add your security admin (`myAdmin`) to the list.
2. Delete the asterisk (*), which made everyone a security administrator.



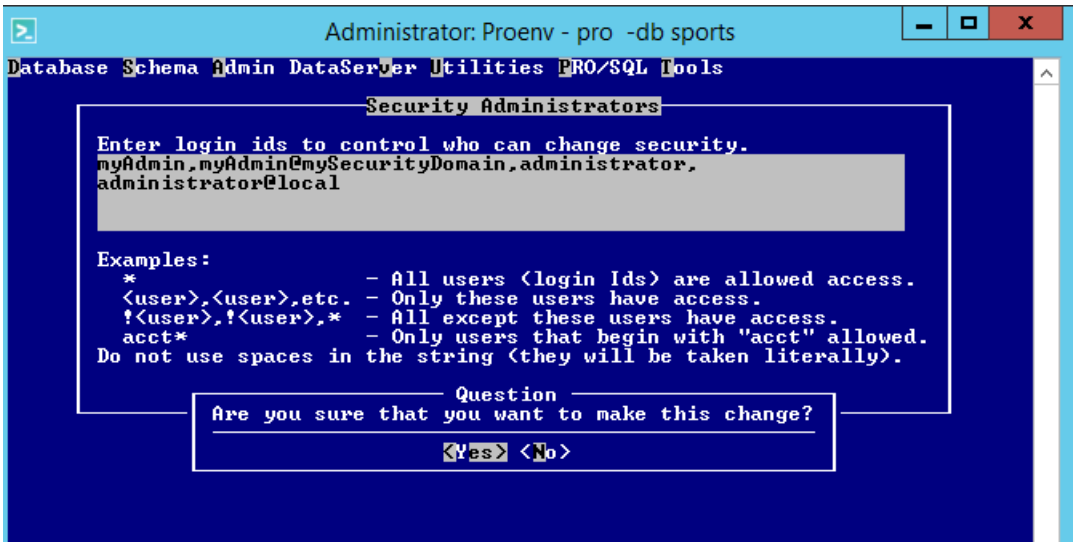
3. Edit the list to explicitly name the following users:

```
myAdmin,myAdmin@mySecurityDomain,administrator
administrator@local
```

There cannot be spaces between Security Administrators users, but you can have a linebreak to ensure the user entries display on the screen.

Note: You may need to enter your operating system username for your database machine as an administrator to the list. For example, if your local operating system login on the database machine is `hsimpson`, you would need to add `hsimpson` to the list of Security Administrators as well.

4. Press **F1** to save.
5. Select **<Yes>** to make the change.

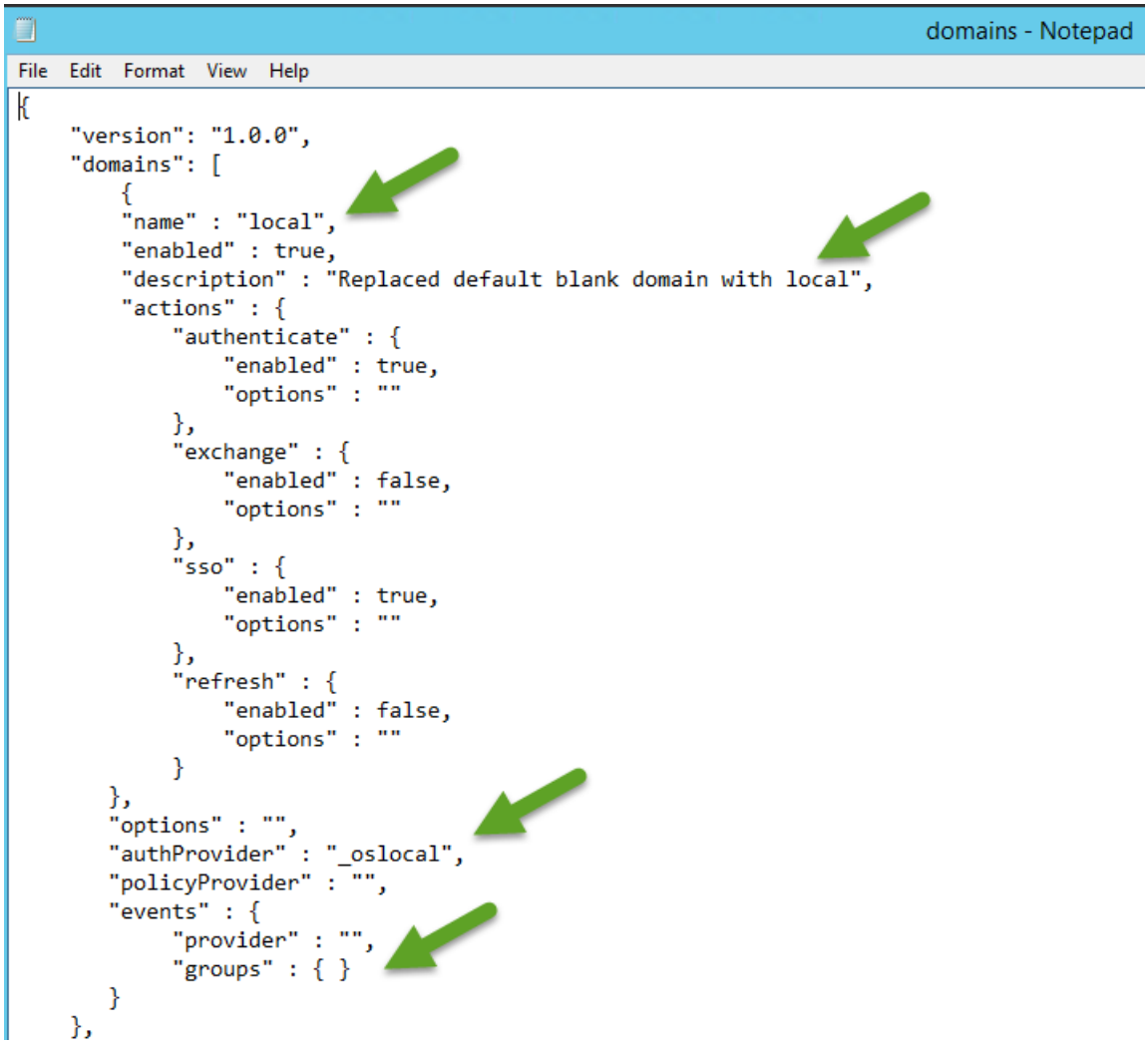


Note: Adding `myAdmin` is for testing purposes only. In a production deployment environment, you would want tight control over who can change security.

Add the local domain to domains.json

In this section, you modify the `domains.json` file to add a local domain on your Authentication Gateway server.

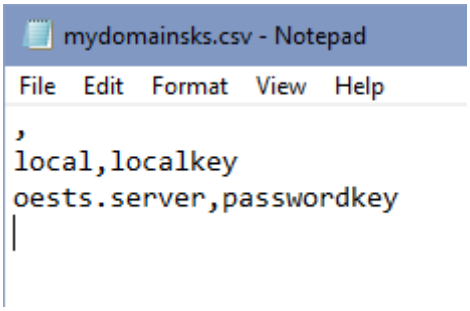
1. On the Authentication Gateway server, open the `C:\OpenEdge\WRK\oeauthserver\webapps\ROOT\WEB-INF\config\domains.json` file in a text editor.
2. Copy the contents of the blank (`"name" : ""`) domain definition, and paste them above or below the original blank domain, so that you have two blank domains in the `domains.json` file.
3. Update the one of the blank domains with the following edits:
   a. Name: **local**
   b. Description: **Local domain for OS credentials**
   c. AuthProvider: **_oslocal**
   d. Groups: **{ }**



Note: There are multiple ways to authenticate through _oslocal. For more information about using _oslocal on Unix-like platforms, see the following Knowledge Base article: OpenEdge authentication gateway(OEAG) fails to authenticate through _oslocal if user account is stored locally in /etc/passwd. If you follow the Knowledge Base article and provide local authentication using `stsclientutil -cmd exchange` requires an additional change to the `domains.json` file:

```
      "exchange":{
         "enabled":true,
         "options":"-processid"
      },
```

4. Save the changes.
5. Open the `mydomainsks.csv` file.
6. Underneath the blank domain (indicated with a comma), enter a line for `local,localkey`:



   Also, confirm that the `oests.server,passwordkey` is correct.

7. Save the changes.
8. Change directory to `C:\OpenEdge\WRK\oeauthserver\webapps\ROOT\WEB-INF\config`:

```
proenv>cd C:\OpenEdge\WRK\oeauthserver\webapps\ROOT\WEB-INF\config
```

9. Run the following command to combine the key and the domain and password files into the required `domains.keystore`:

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

```
proenv>gendomreg mydomainsks.csv domains.keystore
OEDomainRegistryUtil v1.5.3 (01/30/2019)

  [Generate domain registry from inputFile]

input file      : mydomainsks.csv    [CSV]
output file     : domains.keystore    [binary]
```

Note: In production environments, you would remove the `domainsks.csv` file and save it somewhere secure since it can be used to generate the `domains.keystore` file.

10. Stop, clean, and restart the server to apply the changes:

```
proenv>cd C:\OpenEdge\WRK\oeauthserver\bin
proenv>tcman pasoestart -restart
```

11. From your client database machine, test the security token service to verify it will authenticate the user on your machine:

```
proenv>stsclientutil -url https://yourgatewayhostname:8443 -nohostverify -cmd
authenticate -user yourOScredentials@local
Enter user's password:
```

For *yourOScredentials*, enter your operating system login for the Authentication Gateway server machine, and then the password that you use to access the Authentication Gateway machine.

12. You will see the following message if configured correctly:

```
Authentication succeeded
```

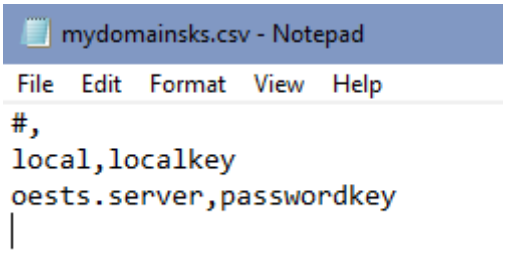
13. Test authentication using the blank domain (such as using the `test` user from the `users.properties` file as from earlier):

```
proenv>stsclientutil -url https://yourgatewayhostname:8443 -nohostverify -cmd
authenticate -user test -password test
```

(Optional) Disable the default blank domain

In the last section, you added the local domain which requires users to be authenticated using the local domain that validates the user based on their log in credentials, but left the blank domain in place, which authenticates users using the `users.properties` file. To consolidates the user credentials to use the operating system credentials, disable the blank domain.

1. On your Authentication Gateway server, open the `mydomainsks.csv` file.
2. Remove or comment-out the blank domain (the line is indicated by a single comma), while leaving the local and oests.server domains, for example:



3. Run the gendomreg command again to combine the updated key and the domain and password files into the required `domains.keystore`:

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

4. From the client machine, test authentication again. First, try to authenticate with the local credentials:

```
proenv>stsclientutil -url https://yourgatewayhostname:8443 -nohostverify -cmd
authenticate -user yourOScredentials@local
Enter user's password:
```

   This should provide the same `Authentication suceeded` message as before.

5. Try to authenticate using the blank domain (such as using the `test` user from the `users.properties` file as from earlier):

```
proenv>stsclientutil -url https://yourgatewayhostname:8443 -nohostverify -cmd
authenticate -user test -password test
```

   Which should provide the following error message:

```
error: stsclientutil: authenticate error: error= 5 (http error)  detail= 400  ()
error: stsclientutil: sts service error:  error= (error_code:2)
error: stsclientutil: sts service error:  error_description= (Domain  is not enabled
 (18737))
Authentication failed. Reason = User authentication failed
```

Disabling the blank domain consolidates the user credentials to use only the local operating system credentials. For testing, however, you can choose not to disable the blank domain. To re-enable the blank domain, simply uncomment the blank domain entry into `mydomainsks.csv` (a line with a comma), and use `gendomreg` to re-generate a `domains.keystore` file. In a production environment, keep your domain registry file`mydomainsks.csv` in a secure location.

Enable the Authentication Gateway on the database

Next, restart the database server and enable the authentication gateway as the enforcer of security:

1. On the database machine, change to the database directory, for example:

```
proenv>cd %WRKDIR%\db
```

2. Ensure your database is not running:

```
proenv>proshut sports2020 -by
```

3. Update the `stsurlutil` by connecting with `-1` to connect to the database using the security administrator and password:

```
proenv>stsurlutil update -url https://yourgatewayhostname:8443 -ssl -nohostverify
 -db sports2020 -1 -U myAdmin@mySecurityDomain -P myAdmin
[2020-08-03T22:32:21.810-04:00] DBSTSURL INFO OPERATION: UPDATE
[2020-08-03T22:32:21.811-04:00] DBSTSURL INFO SESSION:PARAM:
URL:https://yourgatewayhostname:8443,SSL-OPTIONS:-nohostverify
[2020-08-03T22:32:21.811-04:00] DBSTSURL INFO URL: https://yourgatewayhostname:8443
[2020-08-03T22:32:21.811-04:00] DBSTSURL INFO TEST-URL? yes
[2020-08-03T22:32:21.811-04:00] DBSTSURL INFO SSL-OPTIONS: -nohostverify
[2020-08-03T22:32:21.812-04:00] DBSTSURL INFO SERVERNAME:
[2020-08-03T22:32:21.911-04:00] DBSTSURL INFO STS Url created for "sports2020"
[2020-08-03T22:32:21.911-04:00] DBSTSURL INFO OPERATION COMPLETE
```

4. Enable the Authentication Gateway on the database using `proutil`:

```
proenv>proutil sports2020 -C enableauthgateway
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
Warning: It is recommended that connection role security enforcement be configured
when using the OpenEdge Authentication Gateway. (18519)
Authentication Gateway has been enabled for database sports2020. (12479)
```

5. Start the database server with the `-nohostverify` option:

```
proenv>proserve sports2020 -nohostverify
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
22:33:51 BROKER     This broker will terminate when session ends. (5405)
22:33:51 BROKER     The startup of this database requires 40Mb of shared memory.
Maximum segment size is 1024Mb.
22:33:51 BROKER  0: Multi-user session begin. (333)
22:33:51 BROKER  0: Before Image Log Initialization at block 5  offset 332. (15321)
22:33:51 BROKER  0: Login by Administrator on CON:. (452)
```

Test the connection to the database

Now that you have added user to the database, added a domain to the database and updated the `domains.json` and `domains.keystore` to hold the new local domain for the Authentication Gateway's Security Token Service you are ready to test the database connectivity.

1. On the database machine, change to the db directory if you are not already there, for example:

```
proenv>cd %WRKDIR%\db
```

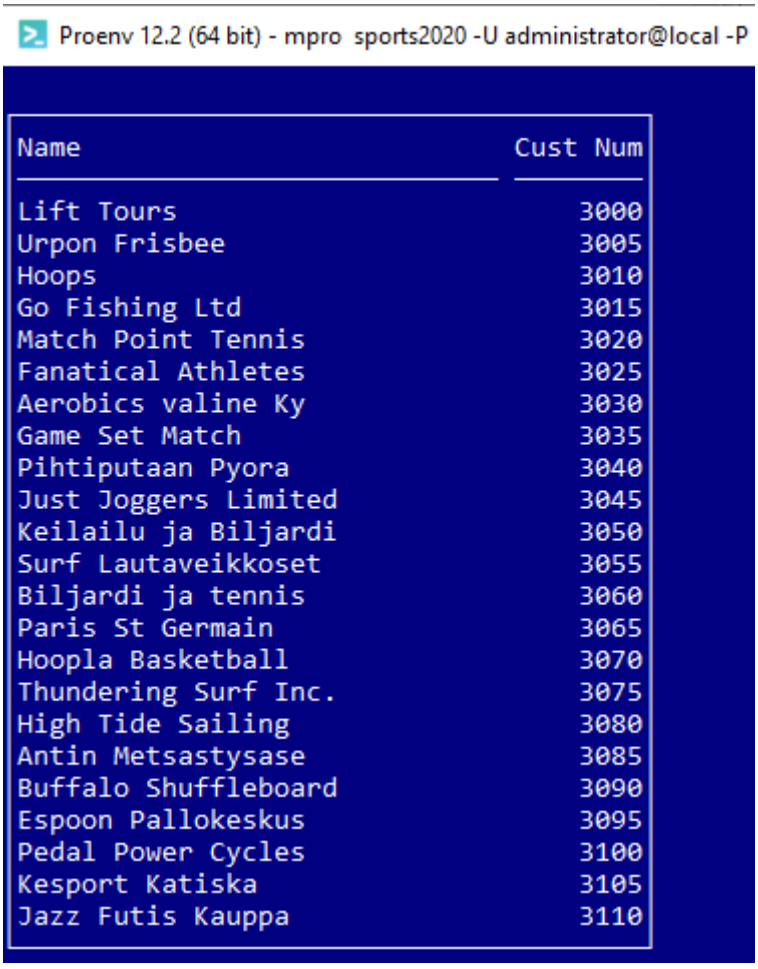
2. Login using the credentials you provided earlier:

```
proenv>mpro sports2020 -U yourOScredentials@local -P yourOSpassword
```

3. After login, you are in a Progress editor and connected to the database, where you can run the code to display database records, such as using the following example code:

```
FOR EACH Customer:
    DISPLAY Name.
    DISPLAY CustNum.
END.
```

4. Press F1 to compile and run the code. It produces an output like the following:



5. Press **ESC** to stop the program.
6. Exit the procedure editor without saving.

7. Test the negative case by trying to connect as an intruder or with the wrong password:

```
proenv>mpro sports2020 -U intruder@local -P "intruder"
```

This produces the error: **"Your Password and Userid intruder do not match"**.

Note: If you kept the blank domain active, you would also be able to access the sports2020 database with the test user and password with a blank domain, for example:

```
proenv>mpro sports2020 -U test -P test
```

---

8. To prove that all logins are handled by the Authentication Gateway, try to log in to your database as the Security Administrator that defined inside your database:

```
proenv>mpro sports2020 -U myAdmin@mySecurityDomain -P myAdmin
```

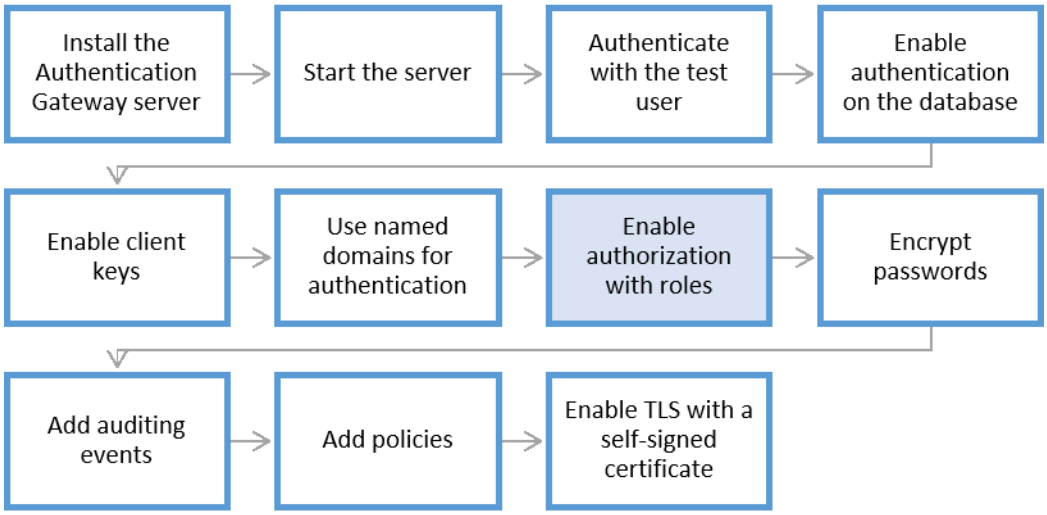
The reason this authentication request fails is because the domain `mySecurityDomain` and user `myAdmin` have only been defined on the database. If you defined a new domain and user on the Authentication Gateway server, and then added the domain to the `domains.json` and `domainsks.csv` file that you use to compile your `domains.keystore`, then you would be able to access your database using these credentials.

Summary

In this topic, we first shut down and disabled the use of the Authentication Gateway on the database. Then we created a security safety net in our database in the form of the `myAdmin@mySecurityDomain` administrative user, which can be used to access the database whenever the Authentication Gateway is disabled. Then we added a local (`_oslocal`) domain to the database using the Database Administrative Tools. In the database, we also defined the security administrative users. Then we added the local domain to the `domains.json` file, and updated the domain registry (`mydomainsks.cvs` and `domains.keystore` files). We added and removed domains (such as the blank domain) to the `mydomainsks.cvs` and regenerated the `domains.keystore` based on domain needs. Then we re-configured out database to use the Authentication Gateway for authentication, and tested the database connection with the local user, which uses the operating system login and password to the Authentication Gateway machine for authentication.

Use connection roles for authorization

In this topic, we will add the connection role access control to the database so that only named and approved users can access the database.



Complete the following steps to implement authorization roles on the database:

- Add system users to the Authentication Gateway machine
- Verify that test user accounts can use the operating system login to access the database
- Enable connection roles for authorization on the database
- Add roles for authorization on the database
- Test roles for authorization on the database

Add system users to the Authentication Gateway machine

There are different rules for adding users to different operating systems. Consult an administrator if you do not have the proper permissions to add users. Assuming you are running on a test system as an Administrator account, you should be able add users according to a standard process that is defined by your specific operating system.

1. Add a few users to the operating system of your Authentication Gateway server machine, for example:

```
Username: testuser1
Password: testuser1

Username: testuser2
Password: testuser2

Username: testuser3
Password: testuser3
```

Verify that test user accounts can use the operating system login to access the database

There should be a few test OS accounts that you defined on the Authentication Gateway server machine, in addition to the standard Administrator account.

Verify each user can login to the database using the local OS domain:

1. (Optional step. The database should already be running and using the Authentication Gateway. Only required if you are unsure about the state of your database.) On the client database machine, ensure your database is pointed at the Authentication Gateway, and that the Gateway is enabled:

```
proenv>cd %WRKDIR%\db

proenv>proshut sports2020 -by
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
Shutdown is executing. (1613)
Shutdown complete. (1614)

proenv>proutil sports2020 -C disableauthgateway
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
fmaFeatureDisable: The feature Authentication Gateway has been disabled. (11175)
Authentication Gateway has been disabled for database sports2020. (12490)

proenv>stsurlutil update -url https://yourgatewayhostname:8443 -ssl -nohostverify
 -db sports2020 -1 -U myAdmin@mySecurityDomain -P myAdmin
[2020-07-30T20:32:30.696-04:00] DBSTSURL INFO OPERATION: UPDATE
[2020-07-30T20:32:30.696-04:00] DBSTSURL INFO SESSION:PARAM:
URL:https://yourgatewayhostname:8443,SSL-OPTIONS:-nohostverify
[2020-07-30T20:32:30.697-04:00] DBSTSURL INFO URL: https://yourgatewayhostname:8443
[2020-07-30T20:32:30.697-04:00] DBSTSURL INFO TEST-URL? yes
[2020-07-30T20:32:30.697-04:00] DBSTSURL INFO SSL-OPTIONS: -nohostverify
[2020-07-30T20:32:30.698-04:00] DBSTSURL INFO SERVERNAME:
[2020-07-30T20:32:30.793-04:00] DBSTSURL INFO STS Url created for "sports2020"
[2020-07-30T20:32:30.794-04:00] DBSTSURL INFO OPERATION COMPLETE

proenv>proutil sports2020 -C enableauthgateway
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
Warning: It is recommended that connection role security enforcement be configured
when using the OpenEdge Authentication Gateway. (18519)
Authentication Gateway has been enabled for database sports2020. (12479)

proenv>proserve sports2020 -nohostverify
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
20:33:06 BROKER     This broker will terminate when session ends. (5405)
20:33:06 BROKER     The startup of this database requires 40Mb of shared memory.
Maximum segment size is 1024Mb.
20:33:06 BROKER  0: Multi-user session begin. (333)
20:33:06 BROKER  0: Before Image Log Initialization at block 10  offset 1083. (15321)
20:33:06 BROKER  0: Login by Administrator on CON:. (452)
```

Note: Start the database using the `-nohostverify` option, because we have not yet implemented valid digital certificates on the Authentication Gateway.

2. From the client database machine, test the database connections of the defined users on your Authentication Gateway server's operating system, for example:

```
proenv>mpro sports2020 -U testuser1@local -P testuser1
proenv>mpro sports2020 -U testuser2@local -P testuser2
proenv>mpro sports2020 -U testuser3@local -P testuser3
proenv>mpro sports2020 -U adminOScredentials@local -P adminOSpassword
```

All valid accounts on the machine have access to the database.

Enable connection roles on the database

Enable connection roles to grant a limited list of users access to the database.

1. On the database machine in the /db directory, use the stsconnroleutil utility with the enable option, along with the credentials of the Administrator account of your Authentication Gateway server:

```
proenv>stsconnroleutil enable -db sports2020 -U adminOScredentials@local -P
adminOSpassword
[2020-07-31T10:48:17.233-04:00] DBCONNROLE INFO OPERATION: ENABLE
[2020-07-31T10:48:17.233-04:00] DBCONNROLE INFO SESSION:PARAM: "
[2020-07-31T10:48:17.233-04:00] DBCONNROLE INFO Role name: _db.connection
[2020-07-31T10:48:17.234-04:00] DBCONNROLE INFO Grantee:
[2020-07-31T10:48:17.280-04:00] DBCONNROLE INFO Connection authorization now enabled
 for "sports2020"
[2020-07-31T10:48:17.281-04:00] DBCONNROLE INFO OPERATION COMPLETE
```

2. Verify that the connection roles are enabled using the status option:

```
proenv>stsconnroleutil status -db sports2020 -U adminOScredentials@local -P
adminOSpassword
[2020-07-31T10:49:32.899-04:00] DBCONNROLE INFO OPERATION: STATUS
[2020-07-31T10:49:32.899-04:00] DBCONNROLE INFO SESSION:PARAM: "
[2020-07-31T10:49:32.900-04:00] DBCONNROLE INFO Status file:
C:\OpenEdge\WRK/dbconnection_status.txt
[2020-07-31T10:49:32.911-04:00] DBCONNROLE INFO OPERATION COMPLETE
```

   This command generates a dbconnection_status.txt file in the OpenEdge %WRKDIR%.

3. Open the C:\OpenEdge\WRK\dbconnection_status.txt to see that the status was enabled:

```
#DbName,Enabled,RoleName
"sports2020",yes,"_db.connection"
```

4. As the user who enabled the use of connection roles, you can grant roles to have access to the database using the grantuser command, for example:

```
proenv>stsconnroleutil grantuser -grantee testuser1@local -can yes -db sports2020 -U
 adminOScredentials@local -P adminOSpassword

[2020-07-31T10:53:35.100-04:00] DBCONNROLE INFO OPERATION: GRANT
[2020-07-31T10:53:35.100-04:00] DBCONNROLE INFO SESSION:PARAM: GRANTEE:testuser1@local
[2020-07-31T10:53:35.101-04:00] DBCONNROLE INFO Grantee: testuser1@local
[2020-07-31T10:53:35.101-04:00] DBCONNROLE INFO Can-grant? no
[2020-07-31T10:53:35.129-04:00] DBCONNROLE INFO Connection authorization now granted
 for user "testuser1@local" on db "sports2020"
[2020-07-31T10:53:35.129-04:00] DBCONNROLE INFO OPERATION COMPLETE
```

Note: The -can option is for the priviledge to grant other users, not the connection role itself. Use of the -can switch should be limited to database administrators.

5. You can verify the list of grantees with the list option:

```
proenv>stsconnroleutil list -db sports2020 -U adminOScredentials@local -P
adminOSpassword
[2020-07-31T10:54:47.700-04:00] DBCONNROLE INFO OPERATION: LIST
[2020-07-31T10:54:47.701-04:00] DBCONNROLE INFO SESSION:PARAM: "
[2020-07-31T10:54:47.701-04:00] DBCONNROLE INFO List file:
```

```
C:\OpenEdge\WRK122/dbconnection_list.txt
[2020-07-31T10:54:47.729-04:00] DBCONNROLE INFO OPERATION COMPLETE
```

This command produces a `dbconnection_list.txt` file in the OpenEdge `%WRKDIR%`.

6. Open the `C:\OpenEdge\WRK\dbconnection_list.txt` to see the authorized roles:

```
#DbName,Grantee,CanGrant,Grantor,RoleName
"sports2020","testuser1@local",no,"administrator@local","_db.connection"
"sports2020","administrator@local",yes,"administrator@local","_db.connection"
```

7. Try logging into the database with the credentials of the user you granted authorization to:

```
proenv>mpro sports2020 -U testuser1@local -P testuser1
```

8. Try logging into the database with another user, who you haven't yet authorized to use the database:

```
proenv>mpro sports2020 -U testuser2@local -P testuser2
```

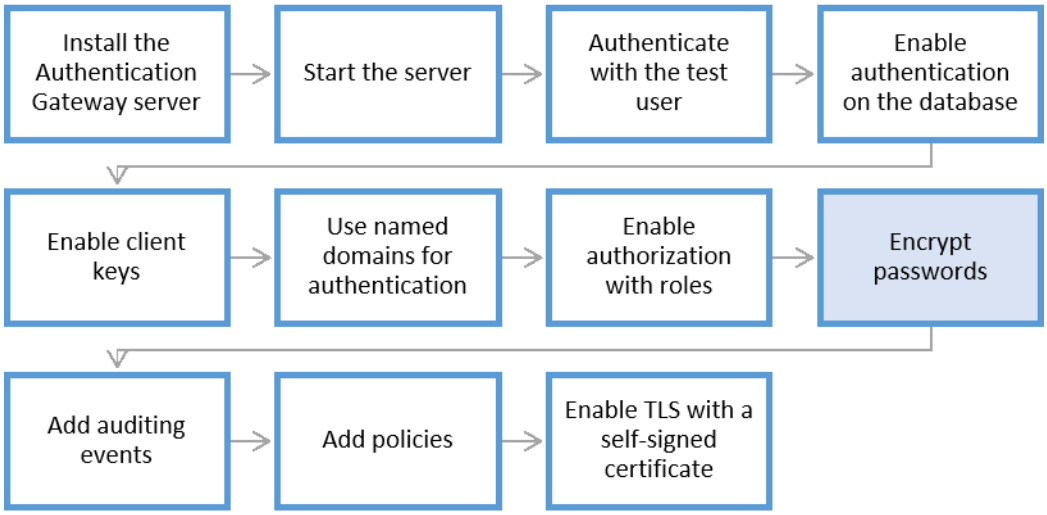
This produces a "password and user ID do not match" error. You could grant this user authorization using the `stsconnroleutil grantuser` command that was previously described in this topic.

Summary

In this topic, you enabled the use of connection roles to add a layer of authorization on top of the authentication of users. The stsconnroleutil has options to `enable` the functionality, `grantuser` to name users, and `status` to check the status of the enabled roles. Once you have established a valid list of connections roles you can then write custom code to limit access once authenticated and authorized for access to the database.

Use encrypted passwords in the sts.properties file

We currently have passwords in the `sts.properties` file that are in clear text. In this topic, we will encode those passwords using either the genpassword or the stspwdutil utility that are available with the OpenEdge Authentication Gateway.



Creating encoded passwords

To encode the password for `sts.server.keyfilepassword=password` found in the `oeauthserver\webapps\ROOT\WEB-INF\config\sts.properties` file:

1. Verify the password for `sts.server.keyfilepassword` in the `sts.properties` file. It should read as `password`.
2. Generate the encoded value of "password" using the genpassword utility, for example:

```
proenv>genpassword -password password
20333c34252a2137
```

Note: As of OpenEdge 12.1, the genpassword utility supports stronger password encoding algorithms by specifying the `-prefix` option in the command. It is recommended that you use a stronger enconding method than the default. For more information, see 'genpassword' in *Manage Keys and Certificates*. The following example shows the encoding output using one of the many supported encoding algorithms:

```
proenv>genpassword -prefix ae1h1 -password password
ae1h1::863afa19e5aab95da11f66c684fb1bda9d858cfb828dec0b0b5d6cf42fb68915
```

3. Modify your `sts.properties` file using a text editor, adding `oech1::` before the output from the genpassword command, for example:

```
sts.server.keyfilepassword=oech1::20333c34252a2137
```

Note: If you used a stronger encoding algorithm, the prefix to the encoded password is different from `oech1::`. An alternate prefix is included in the output of the genpassword command, which should be used in the same manner as the example `keyfilepassword` value above.

4. Stop, clean, and restart the Authentication Gateway server:

```
proenv>cd C:\OpenEdge\WRK\oeauthserver\bin
proenv>tcman pasoestart -restart
```

5. On your database machine, verify you can still log into the database using the local operating system credentials of the Authentication Gateway server, for example:

```
proenv>cd %WRKDIR%\db
proenv>mpro sports2020 -U adminOScredentials@local -P adminOSpassword
```

Creating encrypted password for LDAP configuration

The `genpassword` utility is a password encoding provider for OpenEdge products. When LDAP is used as the third-party authentication source, you need to use a Java standard that LDAP can understand, which is provided by the `stspwdutil` tool.

Although we don't have LDAP configured in this workshop (yet), we will go through the steps that can be used to encrypt the `sts.ldap.context.password` that can be used with an LDAP server:

1. Verify the password for `sts.ldap.context.password=secret` in the `sts.properties`. It should read "secret", as that is the general default password for LDAP servers.
2. Generate the encrypted value using stspwdutil:

```
proenv>stspwdutil encrypt secret
oeph0::C7580909560800DBCBD60808A618A1B6
```

3. The stspwdutil tool can also have a key pad ID of 0 – 9. If no value is set, the default is 0. For example, using a key pad ID of 5:

```
proenv>stspwdutil encrypt secret 5
oeph5::8ADD701874613EE778988A41E6EB26DE
```

4. Modify the ldap context password in the `sts.properties` file with one of the values you generated, for example:

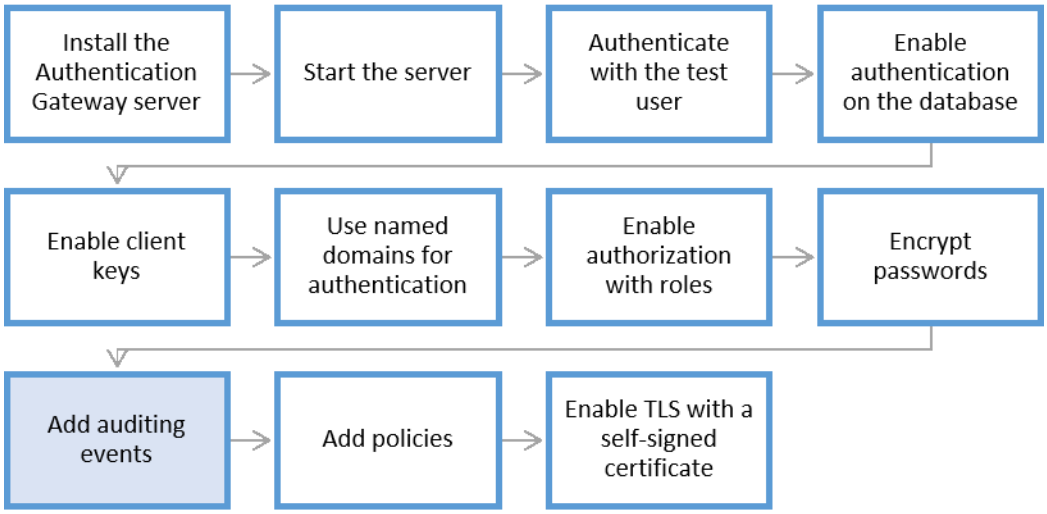
```
sts.ldap.context.password=oeph5::8ADD701874613EE778988A41E6EB26DE
```

Summary

In this topic, we replaced all clear text passwords in the `sts.properties` file with either encoded (using `genpassword`) or encrypted (using `stspwdutil`) values.

Add auditing to the OpenEdge Authentication Gateway server

In this topic, you enable the OpenEdge Authentication Gateway to monitor the database for login events, and add ABL code to the Authentication Gateway that records login attempt information to the Authentication Gateway agent log file.



The built-in mechanism for OpenEdge Authentication Gateway is “Events”. Using events, you can write ABL code that is run whenever a specific event is fired. This is done by modifying your `domains.json` file to set events groups to `true`, and assigning a class file to be run as the `eventProvider`.

In this topic, you use an ABL class file to write events to the `oeauthserver.agent.log` file when a login event occurs.

Add event auditing

Modify your `domains.json` to use a specific `eventsProvider`, and fire events on specific groups.

1. Open the `C:\OpenEdge\WRK\oeauthserver\webapps\ROOT\WEB-INF\config\domains.json` file in an editor.
2. Make the following changes to the local domain:
   - Add a `login` events provider.
   - Change all of the groups from `false` to `true` in the `events` section.
   - Add an `eventPolicy` of `login` to the bottom of the local domain definition.

   These edits are shown in bold in the following local domain definition:

```
{
     "name" : "local",
        "enabled" : true,
        "description" : "Replaced default blank domain with local",
        "actions" : {
            "authenticate" : {
```

```
            "enabled" : true,
            "options" : ""
           },
           "exchange" : {
            "enabled" : false,
            "options" : ""
           },
           "sso" : {
            "enabled" : true,
            "options" : ""
           },
           "refresh" : {
            "enabled" : false,
            "options" : ""
           }
      },
      "options" : "",
      "authProvider" : "_oslocal",
      "policyProvider" : "",
      "events" : {
           "provider" : "login",
           "groups" : {
               "client" : true,
               "tokenAuthenticate" : true,
               "tokenExchange" : true,
               "tokenRevoke" : true,
               "tokenValidate" : true,
               "tokenRefresh" : true,
               "policy" : true,
               "sessionLogin" : true,
               "sessionLogout" : true,
               "sessionValidate" : true,
               "sessionRefresh" : true
           }
      },
  },
```

3. At the bottom of the `domains.json` file, edit the `eventProviders` section to add a `login` event, as well as information about finding the SampleEventHandler and hash information (which can be blank), for example:

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

Note: For testing purposes, the hash information can be blank or omitted. However, the hash value is a security feature to ensure that the correct code is being run. For more information about setting hash values for events and policies, see Configure event callbacks on page 88.

4. On the Authentication Gateway machine, create the directory path for the `SampleEventHandler` code so that it is added to the PROPATH:

```
proenv>cd %WRKDIR%\oeauthserver\openedge
proenv>mkdir com\progress\sts
proenv>cd com\progress\sts
```

5. Create the `SampleEventHandler.cls` file in the newly-created `oeauthserver\openedge\com\progress\sts` directory using the following code:

```
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
 "C-P Domain: " poPrincipal:Token:domain-name skip
 "C-P Roles: " poPrincipal:Token:roles skip
 "C-P user id: " poPrincipal:Token:user-id skip
    "context:" poDomainCtx.
 MESSAGE "       ".
    end method.
end class.
```

6. Stop, clean, and, restart the Authentication Gateway server:

```
proenv>cd %WRKDIR%\oeauthserver
proenv>bin\tcman pasoestart -restart
```

7. Use your database machine to connect to the database using an authorized local user, for example:

```
proenv>cd %WRKDIR%\db
proenv>mpro sports2020 -U testuser1@local -P testuser1
```

8. Open the `C:\OpenEdge\WRK\oeauthserver\logs\oeauthserver.agent.log` and view the bottom entries, which look like the following:

```
2020-08-01T15:11:11.194-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) sender:  STS
2020-08-01T15:11:11.194-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) event:  CLIENT-AUTHENTICATING
2020-08-01T15:11:11.194-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Token:  1251
2020-08-01T15:11:11.194-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Domain:  local
2020-08-01T15:11:11.194-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Roles:
2020-08-01T15:11:11.194-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P user id:  testuser1
2020-08-01T15:11:11.194-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) context:
Progress.Json.ObjectModel.JsonObject_1253
2020-08-01T15:11:11.195-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:18)
2020-08-01T15:11:11.360-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) sender:  STS
2020-08-01T15:11:11.360-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) event:  CLIENT-AUTHENTICATED
2020-08-01T15:11:11.360-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
```

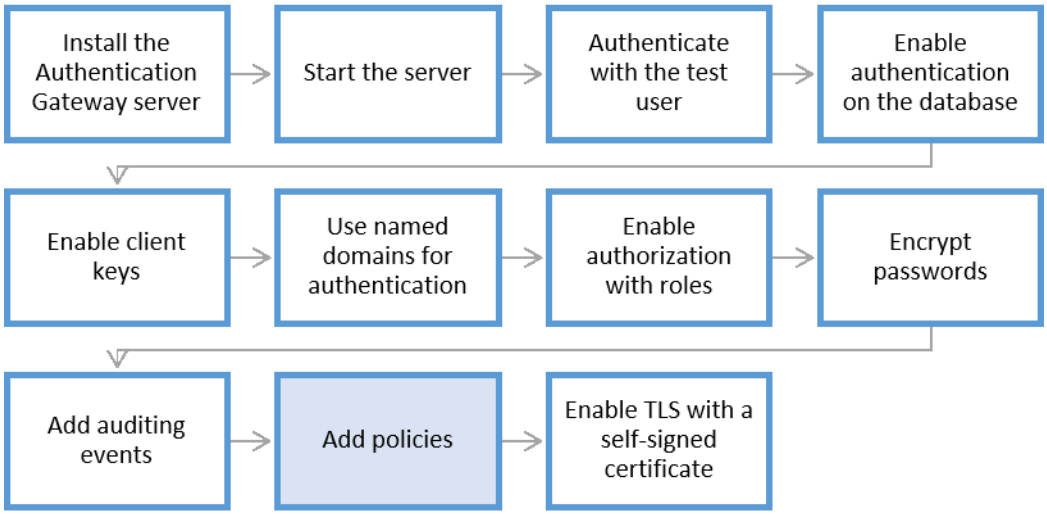
```
 com.progress.sts.SampleEventHandler' Line:11) C-P Token:  1258
2020-08-01T15:11:11.360-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Domain:  local
2020-08-01T15:11:11.360-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Roles:
2020-08-01T15:11:11.360-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P user id:  testuser1
2020-08-01T15:11:11.360-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) context:
Progress.Json.ObjectModel.JsonObject_1260
2020-08-01T15:11:11.360-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:18)
```

Summary

In this topic, we configured events to be recorded in the agent log file. This provides an audit trail for Authentication Gateway administrators to have a view of login events in the server's agent log file.

Add policies to the OpenEdge Authentication Gateway server

In this topic, we will implement a policy on the Authentication Gateway server to disallow users, except "Administrator", from accessing the OpenEdge database on the weekend.



Policies allow you to further manage authorization in the OpenEdge Authentication server. You may need to implement different policies for different users, roles, applications, or geographies, depending on your business needs. This topic shows how to setup a simple policy.

In this topic, we will add a policy provider to a domain and then set the ABL class to run the policy.

Add event auditing

Modify your `domains.json` to use a specific `policyProvider` to be run for that domain:

1. Open the `C:\OpenEdge\WRK\oeauthserver\webapps\ROOT\WEB-INF\config\domains.json` file in an editor.
2. For the local domain, set the `policyProvider` to `"weekend"`, for example:

```
"authProvider":"_oslocal",
        "policyProvider":"weekend",
```

3. At the bottom of the `domains.json` file, edit the `policyProviders` section to add a `weekend` policy, and information about finding the Weekend Policy and hash information (which can be blank), for example:

```
   "policyProviders":{
      "weekend":{
         "type":"example.sts.WeekendPolicy",
         "hash":""
      }
   },
```

Note: For testing purposes, the hash information can be blank or omitted. However, the hash value is a security feature to ensure that the correct code is being run. For more information about setting hash values for events and policies, see Configure event callbacks on page 88.

4. Create the directory path for the WeekendPolicy code so that it is added to the PROPATH:

```
proenv>cd %WRKDIR%\oeauthserver\openedge
proenv>mkdir example\sts
```

5. Create the `WeekendPolicy.cls` file in the newly-created `oeauthserver\openedge\example\sts` directory using the following code:

```
using OpenEdge.Security.PAMStatusEnum.
using OpenEdge.Security.Principal.
using OpenEdge.Security.STS.IPolicyProvider.
using Progress.Json.ObjectModel.JsonObject.

class example.STS.WeekendPolicy implements IPolicyProvider:
    method public PAMStatusEnum ApplyPolicy( input pcSender as character,
                                             input pcPolicyName as character,
                                             input poPrincipal as Principal,
                                             input poDomainCtx as JsonObject,
                                             output pcStatusDetail as character ):

        // Allow user 'Administrator' to login every day of the week
        if    poPrincipal:Token:user-id <> 'Administrator'
           then
        do:
            // Only allow logins on weekdays for users not named 'Administrator'
            if    weekday(now) eq 1 //SUNDAY
               or weekday(now) eq 7 //SATURDAY
               then
            do:
                // Record the policy violation
               log-manager:write-message(substitute('Weekend login attempt detected
 by &1',
                                                    poPrincipal:Token:qualified-user-id),

                                                 'POLICY':u).
                // Don't allow the login
```

```
                return PAMStatusEnum:LoginDenied.
            end.
        end.

        return PAMStatusEnum:Success.
    end method.

end class.
```

6. Stop, clean, and restart the Authentication Gateway server:

```
proenv>cd %WRKDIR%\oeauthserver
proenv>bin\tcman pasoestart -restart
```

7. On the database machine, connect to the database:

```
proenv>cd %WRKDIR%\db
proenv>mpro sports2020 -U testuser1@local -P testuser1
```

If you attempt to login with a user not named Administrator on the weekend, this login attempt returns a "Password and Userid do not match" error.

Note: If you don't want to wait for the weekend to test this policy, edit the `WeekendPolicy.cls` file to set `weekday(now)` equal to whichever day of the week it currently is: `1` being Sunday, and `7` being Saturday.

8. Use an editor to check the `oeauthserver.agent.log` for the policy message:

```
2020-08-01T14:22:11.292-0400 002324 005372 1 AS-14 ?:?:? POLICY        Weekend login
 attempt detected by testuser1@local
```

9. However, the Administrator can still log in on the weekends. Assuming you are running this exercise on a weekend, or have modified the `WeekendPolicy.cls` file appropriately, try to log in to the database with the local operating system Administrator user:

```
proenv>mpro sports2020 -U Administrator@local -P adminOSpassword
```

Note: If the local Administrator user on your Authentication Gateway machine uses a different user ID than 'Administrator', edit the `WeekendPolicy.cls` file to let that named user log in any day of the week.

The Policy warning statement does not print to the agent log file, but some information is printed to the log file because of the event auditing that was added to the domain, such as the different events that occur during the Administrator policy authorization:

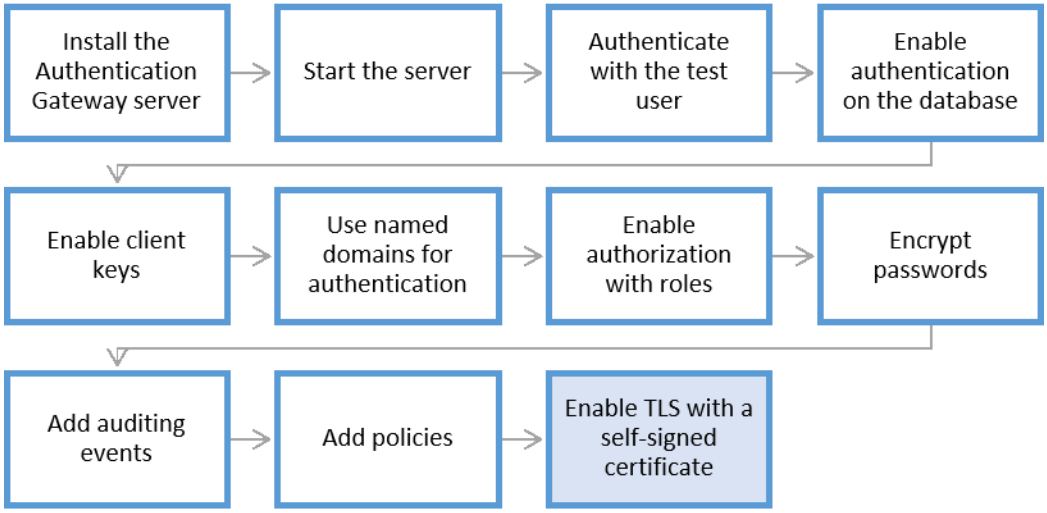
```
2020-08-01T15:05:46.964-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) sender:  STS
2020-08-01T15:05:46.964-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) event:  CLIENT-AUTHENTICATING
2020-08-01T15:05:46.964-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Token:  1215
2020-08-01T15:05:46.964-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Domain:  local
2020-08-01T15:05:46.964-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Roles:
2020-08-01T15:05:46.964-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P user id:  Administrator
2020-08-01T15:05:46.964-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) context:
Progress.Json.ObjectModel.JsonObject_1217
2020-08-01T15:05:46.964-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:18)
2020-08-01T15:05:47.023-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) sender:  STS
2020-08-01T15:05:47.023-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) event:  POLICY-APPLYING
2020-08-01T15:05:47.023-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Token:  1222
2020-08-01T15:05:47.023-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Domain:  local
2020-08-01T15:05:47.023-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Roles:  ROLE_Administrators
2020-08-01T15:05:47.023-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P user id:  Administrator
2020-08-01T15:05:47.023-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) context:
Progress.Json.ObjectModel.JsonObject_1224
2020-08-01T15:05:47.023-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:18)
2020-08-01T15:05:47.030-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) sender:  STS
2020-08-01T15:05:47.030-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) event:  POLICY-APPLIED
2020-08-01T15:05:47.030-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Token:  1237
2020-08-01T15:05:47.030-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Domain:  local
2020-08-01T15:05:47.030-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Roles:  ROLE_Administrators
2020-08-01T15:05:47.030-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P user id:  Administrator
2020-08-01T15:05:47.030-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) context:
Progress.Json.ObjectModel.JsonObject_1239
2020-08-01T15:05:47.030-0400 013096 010064 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:18)
2020-08-01T15:05:47.211-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) sender:  STS
2020-08-01T15:05:47.211-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) event:  CLIENT-AUTHENTICATED
2020-08-01T15:05:47.211-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Token:  1244
2020-08-01T15:05:47.211-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Domain:  local
2020-08-01T15:05:47.211-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P Roles:
2020-08-01T15:05:47.211-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) C-P user id:  Administrator
2020-08-01T15:05:47.211-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:11) context:
Progress.Json.ObjectModel.JsonObject_1246
2020-08-01T15:05:47.212-0400 013096 011228 1 AS-15 ?:?:? -- (Procedure: 'RecordEvent
 com.progress.sts.SampleEventHandler' Line:18)
```

Summary

In this topic, we configured policy that did not allow connection to the database on the weekend unless a user is named as "Administrator". We also saw some of the advanced logging that occurs when policy events take place.

Replace the default key and certificate on OpenEdge Authentication Gateway server

The OpenEdge Authentication Gateway server is accessed by encrypted HTTPS calls. To learn the basic setup and review the configuration of the server, we have been suppressing the error messages with the `-nohostverify` options on the server and test utilities.



To fully implement an OpenEdge Authentication Gateway Server, the server must always require a valid certificate. In this topic, we implement a self-signed certificate for our server, and copy it to the client machines that access the Authentication Gateway server.

In a production environment, you would need to submit a public key to a Certificate Authority (CA) to request a server certificate, and then copy the root and server certificate to your Authentication Gateway server and client machines. To request a certificate, you would first create a Key Pair, which is a private key keystore (a `.pk1` file) and public key (a `.pk10` file), and then submit the `.pk10` file to the CA.

For more information about this process, see 'Obtain a certificate from a certificate authority' in *Manage PAS for OpenEdge*. This activity is out of the scope for this tutorial.

This topic covers the following steps:

1. Create a self-signed certificate on the Authentication Gateway server
2. Import the certificate on the Authentication Gateway server
3. Copy certificate to the client machine (or machines)
4. Configure the Authentication Gateway server to use the keystore repository
5. Test the connections

Create a self-signed certificate on the Authentication Gateway server

To create a self-signed SSL certificate with OpenEdge for use with Authentication Gateway clients:

1. In Proenv on the Authentication Gateway server, navigate to the `oeauthserver\conf` directory
2. Rename the original `tomcat-keystore.p12` file to `tomcat-keystore.p12.orig` in case you want to revert to the default keystore:

```
proenv>rename tomcat-keystore.p12 tomcat-keystore.p12.orig
```

Note: In this case, we are simply renaming the original `tomcat-keystore.p12` file, rather than creating a backup copy of the file. This is because the next steps would add an additional entry to the existing `tomcat-keystore.p12` file if the `tomcat-keystore.p12` file were present in the `oeauthserver\conf` directory.

3. Use the `keytool` utility to generate a key pair keystore file called `tomcat-keystore.p12`:

```
proenv>keytool -genkeypair -keyalg RSA -alias oeAuth -keystore tomcat-keystore.p12
-storetype pkcs12 -validity 100 -keysize 2048 -dname
"CN=gatewayhostname,OU=Training,O=Progress,L=Bedford,ST=Massachusetts,C=US"
```

For the `-dname` option, provide the Authentication Gateway hostname as the CN, and then fill in the rest of the Domain Name with attributes that best suit your organization. This command produces the `tomcat-keystore.p12` in the `oeauthserver\conf` directory.

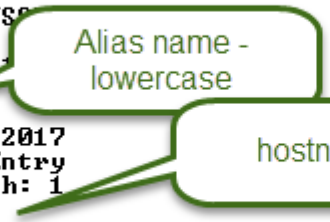
Provide a password for the keystore, in this example you can use `password` with the usual caveats.

4. (Optional) Verify the contents of the keystore using the password you created in the last step:

```
proenv>keytool -list -v -keystore tomcat-keystore.p12 -storetype pkcs12
Enter keystore password:
```

Review the output, which contains the one **oeauth** alias and additional information about the keystore, as seen in the following screen capture:

```
proenv>%DLC%\jdk\bin\keytool -list -v -keystore tomcat-keystore.p12 -storetype p
kcs12
Enter keystore password:

Keystore type: PKCS12
Keystore provider: SunJS

Your keystore contains

Alias name: oeauth
Creation date: Apr 19, 2017
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=WIN-06DDMV5DH5S, OU=Training, O=Progress, L=Bedford, ST=MA, C=US
Issuer: CN=Application Server Test CA, OU=ITT, OU=OpenEdge, O=Progress Software
Corp, DC=PSCProgress, DC=com
Serial number: 516591340000000000038
Valid from: Wed Apr 19 18:35:55 UTC 2017 until: Fri Apr 19 18:35:55 UTC 2019
Certificate fingerprints:
```



5. Export the certificate as a file called `selfsigned.cer` using the keystore password you created in a previous step:

```
proenv>keytool -exportcert -rfc -alias oeAuth -keystore tomcat-keystore.p12 -file
selfsigned.cer
Enter keystore password:
Certificate stored in file <selfsigned.cer>
```

This certificate is also created in the `oeauthserver\conf` directory.

Import the certificate into the Authentication Gateway server

Use the certutil command to make certificate files in the `certs` directory. These files are used by clients who require the necessary keys when connecting.

1. On the Authentication Gateway machine, run the `certutil -import` command to import the `selfsigned.cer` and create a certificate store entry with a generated alias name in the `%DLC%\certs` directory:

   ```
   proenv>certutil -import selfsigned.cer
   Importing trusted certificate to alias name: dd5d2225
   ```

   The certificate alias name will likely be different on your system.

2. Verify that the `certutil -import` command created a file with the extension `.0` in the `%DLC%\certs` directory. In this example, the file would be called `dd5d2225.0`.

Copy the certificates to the client machine (or machines)

Every client (database or stsclientutil user) that will connect to the Authentication Gateway server needs a copy of the certificates that were generated by the `certutil -import` commands.

1. Copy the newly-created `.0` file in the `%DLC%\certs` directory of the Authentication Gateway server to the `%DLC%\certs` directory of each OpenEdge client that will authenticate against the Authentication Gateway.

   In the above example, the `certutil -import` generated a server certificate named `dd5d2225.0` in the `%DLC%\certs` directory of the Authentication Gateway server. Copy this file to your client database machine into the `%DLC%\certs` directory. Repeat this process for each client machine that uses the Authentication Gateway, if you have multiple machines set up.

Configuring the Authentication Gateway server to use the keystore

If you used a new keypass or keyalias you will need to update the `catalina.properties` file, so that your Authentication Gateway server will have the correct keypass and keyalias to use the updated keystore.

1. On the Authentication Gateway server machine, open the `C:\OpenEdge\WRK\oeauthserver\conf\catalina.properties` file in a text editor.
2. Search for the `# JSSE keystore` section.

   ```
   # JSSE keystore used by server.xml for its server key & certificates
   psc.as.https.keypass=password
   psc.as.https.keyalias=test
   psc.as.https.storeType=PKCS12
   ```

3. Edit the keypass and key alias to match your system. You are replacing the alias for the default certificate that is shipped with OpenEdge. This example uses the values created for this exercise. In a production system you would be customizing this like all other passwords and replacement files. Note that the sslc utility lowercased the alias name in the Tomcat keystore, if you do not match what was shown in the listing (lowercased `oeauth`) you will fail to connect.

- `psc.as.https.keypass=password`
- `psc.as.https.keyalias=oeauth`

```
# JSSE keystore used by server.xml for its server key & certificates
psc.as.https.keypass=password
psc.as.https.keyalias=oeauth
psc.as.https.storeType=PKCS12
```

4. Save and close the file.

Test the connection

In this section, you will restart the server to get the appropriate credentials from the keystore to be able to participate in secured communication. The result is that you will be able to connect without using the `-nohostverify` option, because all participants have the appropriate keystores and certificate for communicating.

1. Stop, clean and restart the Authentication Gateway server from the `oeauthserver` directory:

```
proenv>cd %WRKDIR%\oeauthserver
proenv>bin\tcman pasoestart -restart
```

2. On your client machine, use `stsclientutil` to ping the Security Token Service for authentication. Be sure to replace *`yourgatewayhostname`* in the following command. This time you will not need the `-nohostverify` option because you have a valid certificate.

```
proenv>stsclientutil -url https://yourgatewayhostname:8443 -cmd ping
ping ... OK
```

3. On your client machine, use stsclientutil to pass a set of credentials to the Authentication Gateway for authentication. Be sure to replace *`yourgatewayhostname`* in the following command. Again, these client calls to the Authentication Gateway server no longer need the `-nohostverify` option.

```
proenv>stsclientutil -url https://yourgatewayhostname:8443 -cmd authenticate -user
 testuser1@local -password testuser1
Authentication Succeeded

proenv>stsclientutil -url https://yourgatewayhostname:8443 -cmd authenticate -user
 test -password test
Authentication Succeeded
```

Note: Ensure you have the correct domains enabled on your Authentication Gateway to test your authentication with the `test` user. In order for the `test` user and password to be authenticated, the blank domain must still be enabled on the Authentication Gateway server. If you have replaced the blank domain with a local domain, as in the Use a named domain for authentication on page 39 exercise, you can only test the authentication with the local OS credentials (such as the `testuser1@local` that you added to your Authentication Gateway server, or *`yourOScredential`*`@local` and *`yourOSpassword`*) for your Authentication Gateway server.

---

4. Test an authentication attempt with a invalid user, for example:

```
proenv>stsclientutil -url https://yourgatewayhostname:8443 -cmd authenticate -user
 intruder -password intruder
```

```
error: stsclientutil: authenticate error: error= 5 (http error)  detail= 400  ()
error: stsclientutil: sts service error:  error= (error_code:2)
error: stsclientutil: sts service error:  error_description= (Bad credentials)
Authentication failed. Reason = User authentication failed
```

5. On the client machine, stop the database:

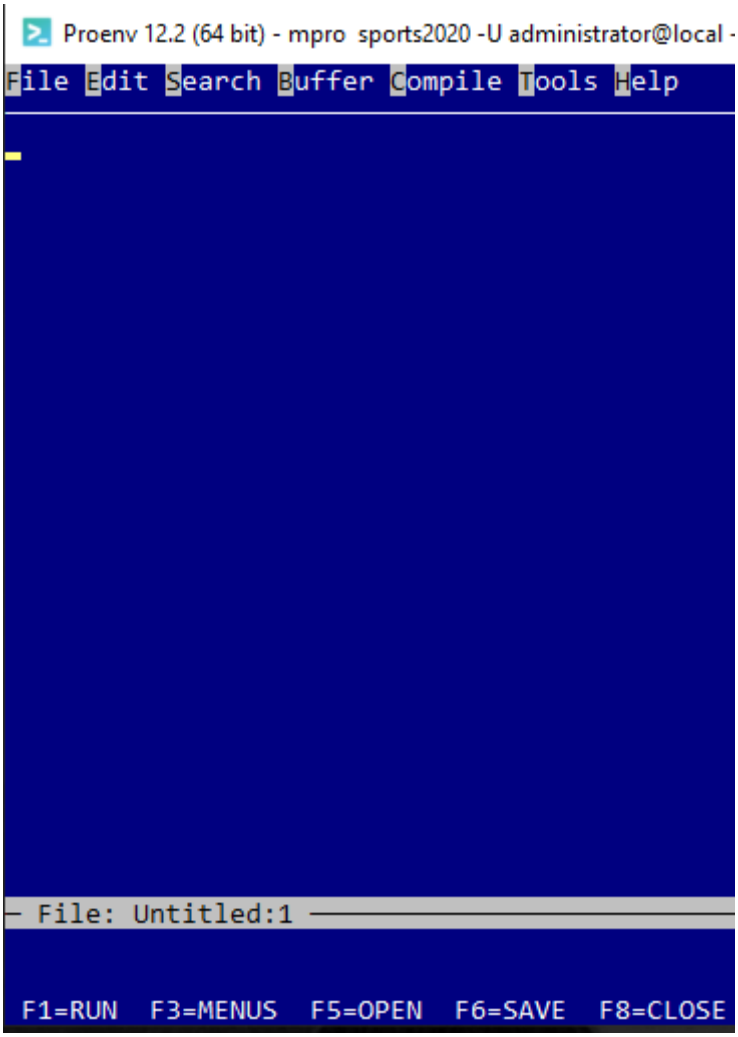
```
proenv>cd %WRKDIR%\db

proenv>proshut sports2020 -by
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
Shutdown is executing. (1613)
Shutdown complete. (1614)
```

6. Restart the database without `-nohostverify`:

```
proenv>proserve sports2020
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
15:07:30 BROKER     This broker will terminate when session ends. (5405)
15:07:30 BROKER     The startup of this database requires 40Mb of shared memory.
Maximum segment size is 1024Mb.
15:07:30 BROKER  0: Multi-user session begin. (333)
15:07:30 BROKER  0: Before Image Log Initialization at block 5  offset 5143. (15321)
15:07:30 BROKER  0: Login by Administrator on CON:. (452)
```

7. Test the database connection without using the `-nohostverify` option:

```
proenv>mpro sports2020 -U adminOScredentials@local -P adminOSpassword
```



Summary

In this topic, you implemented certificates for the Authentication Gateway server and client OpenEdge database. You did all this so the HTTPS connections would work without the `-nohostverify` and the clients, server, and database could communicate using valid certificates encrypting the information passed over your connections.

3

Configure the OpenEdge Authentication Gateway

The OpenEdge Authentication Gateway is a Progress Application Server for OpenEdge (PAS for OpenEdge) instance where the OESTS web application (`oests.war`) is deployed, rather than the standard OEABL (`oeabl.war`) web application. It functions as a Security Token Service (STS) for database, server, and application connections that returns a validated Client-Principal object in response to client requests for database connections.

The Authentication Gateway is a secured HTTPS server that provides STS functionality, which includes user authentication and client-principal generation. The use of a common Authentication Gateway for PAS for OpenEdge instances, databases, and applications enhances security and reduces complexity (because, for example, each PAS for OpenEdge server instance does not need to maintain its own copy of OpenEdge Domains and Domain Access-codes).

Connections between a PAS for OpenEdge instance and an Authentication Gateway server are facilitated through the STS AuthenticationProvider, a plug-in to the Progress Application Server's Spring Security framework.

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
- Configure domains
- Configure event callbacks
- Spring Security Configuration
- Create and deploy a certificate for HTTPS with the OpenEdge Authentication Gateway Server

- STS server key configuration
- About STS client key management

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. However, it is not recommended. For more information, see 'Deploy Remote Administration Applications' in *Manage Progress Application Server (PAS) for OpenEdge*.
- Standard PAS for OpenEdge administrator user accounts have been hardened by inserting digested passwords into the *instance-name*`/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 *Manage Progress Application Server (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

Configure 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-name*`/webapps/ROOT/WEB-INF/config/domains.json`, where *instance-name* is the name of the OpenEdge Authentication Gateway 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` as described in Data export utility on page 119.

Additionally there is a *instance-name*`/webapps/ROOT/WEB-INF/config/domains.keystore` file. This is a Java KeyStore (JKS) type file that holds the domain access codes for the configured domains. For more information, see Configure domain access codes on page 81.

The domains.json file

The default `domains.json` file contains the 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-name*`/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 91 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.

Configure 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 OpenEdge 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-name*`/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-name*`/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.
- A practical example of configuring domain access codes is described in the Use a named domain for authentication on page 39 topic in the *OpenEdge Authentication Gateway tutorial*.

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" : {
    }
}
```

Configure 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" />
```

Configure 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.
```

Note: A practical example of adding a policy is described in Add policies to the OpenEdge Authentication Gateway server on page 65.

Configure multi-domain user ID support

In OpenEdge 11.7.5+ and 12.1+, support has been added for multiple fully qualified domain names (FQDN) in OpenEdge Authentication Gateway. Support for domain-qualified user IDs solves an issue for users who perform multi-tenant account lookup and information retrieval: their authentication services are now able to customize access to tenant-aware user account storage, and add tenant specific information to Client-Principal security token fields.

To configure multi-domain user ID support in OpenEdge Authentication Gateway:

1. Open the `domains.json` file for your STS web application. For example, *`oeauthserver-name`*`/webapps/ROOT/WEB-INF/config/domains.json`, where `oeauthserver-name` is the name of the instance where the OESTS web application is deployed.
2. Edit the `domains.json` file to include the FQDN used during the authentication process. For example, if your authentication method is LDAP, and the domain for your application is employee01@progress, add or edit the domain entry to the list:

```
{
      "name" : "PROGRESS",
      "enabled" : true,
      "description" : "LDAP Multi domain authentication",
      "actions" : {
  "authenticate" : {
   "enabled" : true,
   "options" : "-useFQUserIds"
  },
  "exchange" : {
   "enabled" : true,
   "options" : ""
  },
  "sso" : {
   "enabled" : true,
   "options" : ""
  },
  "refresh" : {
   "enabled" : false,
   "options" : ""
  }
      },
      "options" : "",
      "authProvider" : "ldap",
      "policyProvider" : "",
      "events" : {
  "provider" : "",
  "groups" : {}
      }
}
```

The value that allows for multi-domain support is the `-useFQUserIds` switch. Once added, the OpenEdge Authentication Gateway will pass a fully qualified OpenEdge user ID to whichever authentication provider (“authProvider”) is configured.

Note: The authentication providers where the `-useFQUserIds` option can be most useful are OERealm and LDAP because OERealm supports multi-tenant-enabled user accounts, and LDAP can use the fully qualified user ID as an email account name.

The following rules are used by the OpenEdge Security Token Service (STS) web application for constructing and passing a fully qualified user ID to an authentication provider:

1. If the client input contains a blank OpenEdge Domain subfield, the user ID is passed to the authentication provider unaltered. A static OpenEdge Domain name can be configured to be appended to the user ID after a successful authentication.
2. If the client input contains a non-blank OpenEdge domain subfield, and the `-useFQUserIds` option is not specified, then only the user ID subfield is passed to the authentication provider. Upon successful authentication, the client-specified fully qualified user ID is placed into the sealed client-principal.
3. If the client input contains a non-blank OE Domain subfield, and the `-useFQUserIds` option is specified, then the client-specified fully qualified user ID is passed to the authentication provider unaltered. Upon successful authentication, the fully qualified user ID is placed into the sealed client-principal.

The following table shows various client user ID inputs and what values are passed to authentication providers:

Client Specified User ID	Without -useFQUserIds	With -useFQUserIDs	Inserted Into Client-Princpal
user1	user1	user1	user1 [1]
user1@domain1	user1	user1@domain1 [2]	user1@domain1
user1@email@domain1	user1@email [2]	user1@emain@domain1	user1@email@domain1 [2]

Note:

[1] The optional @domain may be blank, a configured static value used by all OpenEdge Domains, or a value derived from one of the user account's granted group/role memberships.

[2] The user1@domain1or user1@email may be passed to an LDAP authentication provider when the user search filter includes matching the LDAP user account's email attribute.

Configure 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.
```

Note: For a practical example of adding events, see Add auditing to the OpenEdge Authentication Gateway server on page 62.

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: The Authentication Gateway is an OpenEdge implementation of a Security Token Service (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="ad" value-ref="adAuthenticationProvider"></b:entry>
  <b:entry key="ldap" value-ref="ldapAuthProvider"></b:entry>
  <b:entry key="oerealm" value-ref="OERealmAuthProvider"></b:entry>
  <b:entry key="_oeusertable" value-ref="OEDBAuthProvider"></b:entry>
  <b:entry key="_oslocal" value-ref="OSLocalAuthProvider"></b:entry>
  <b:entry key="properties" value-ref="PropertiesFileAuthProvider"></b:entry>
    </b:map>
</b:property>
```

This section shows the implementations for the default login providers. 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:

- adAuthenticationProvider — This is an LDAP authentication provider for Microsoft's Active Directory. It is an offshoot of the primary LDAP provider, but limited in scope and functionality specific to Active Directory. It is provided as an alternative to the full LDAP authentication provider.
- ldapAuthProvider — Makes use of the standard spring LDAP authentication provider implementation and requires an LDAP server configuration.
- OERealmAuthProvider — Spring AuthenticationProvider that manages the task of authentication using a user account supplied by an AppServer Realm service. It directly uses the OERealmUserDetailsImpl bean to do the real work and create a Spring UserDetails object that will be placed in a Spring authentication token. For more information about the OERealm Authentication Provider, see About OERealm user account configurations in *Manage Progress Application Server (PAS) for OpenEdge*.
- OEDBAuthProvider — Spring AuthenticationProvider that manages the task of emulating an OpenEdge database's _user account by using an OERealm service. It directly uses the OEDBUserDetails bean to do the real work and create a Spring UserDetails object that will be placed in a Spring authentication token. Performs authentication of login requests using the OERealm ABL interface to an ABL class implementation, but modified for accessing the OpenEdge database user tables.
- OSLocalAuthProvider — Performs authentication of login requests (a client-principal containing a user/password combination) for authentication by the local operating system.

- `PropertiesFileAuthProvider` — Performs authentication of login requests (a client-principal containing a user/password combination) for authentication using the STS web application's `WEB-INF/config/users.properties` file.

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 the 'pkiutil' in *Manage OpenEdge Keys and Certificates* 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.

Note: For a practical example of configuring a certificate for the OpenEdge Authentication Gateway, see Replace the default key and certificate on OpenEdge Authentication Gateway server on page 70.

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).

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

Note: For a practical example of configuring an STS server key and client keys, see Use a server key with the OpenEdge Authentication Gateway on page 30.

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 Configure domains on page 79.)

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-name`*`/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.

Beginning in OpenEdge 12.3, STS client key management can be handled by the Key Distribution service. For more information, see About STS client key management on page 97.

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 `oeauthserver/webapps/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).

Troubleshooting

Check the STS server log file in `oeauthserver/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 `oeauthserver/webapps/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.
```

For more information about log files and logging levels, see 'Log files' in *Learn about Progress Application Server for OpenEdge* and 'Configure logging in PAS for OpenEdge' in *Manage Progress Application Server (PAS) for OpenEdge*.

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.

About STS client key management

OpenEdge 12.3 includes an optional Key Distribution application for the OpenEdge Authentication Gateway that provides remote STS client key management for client OpenEdge installations. If you configure your OpenEdge Authentication Gateway to use an STS server key, each remote OpenEdge installation must generate a corresponding STS client key. The STS client key generation must happen on the client system, and requires the client system to have access to the STS server key. With earlier versions of the OpenEdge Authentication Gateway, this was done though a shared file on a remote drive, or by manually copying the STS server key to the client machine. This was a manual process that could leave the STS server key on disk.

OpenEdge 12.3 offers an STS client key management service that runs on the OpenEdge Authentication Gateway through the Key Distribution application. After you install this service on the OpenEdge Authentication Gateway server using the Key Distribution application, then STS client key generation can be done on the client machine by specifying the URL of the OpenEdge Authentication Gateway server and providing the proper credentials. This can be done through the stskeyutil utility, or through the STSKey plugin for the OpenEdge AdminServer.

The STS client key management service allows system administrators to efficiently manage OpenEdge installations that require an STS client key to access the OpenEdge Authentication Gateway. This service helps administrators avoid the risks of manually creating STS client keys for each OpenEdge installation, and makes it easier to configure remote client machines to use OpenEdge Authentication Gateway.

The STS client key management service is provided by a web application that is packaged as `keydist.war`. The `keydist.war` application runs on the OpenEdge Authentication Gateway server, and can only be deployed to an OpenEdge Authentication Gateway server instance. You cannot deploy the `keydist.war` application to a regular PAS for OpenEdge instance. The `keydist.war` application is distributed in the `$DLC/servers/pasoe/extras` directory.

The `keydist.war` application can be deployed to the OpenEdge Authentication Gateway server using the TCMAN `deploy` command:

```
proenv>oeauthserver/bin/tcman deploy %DLC%/servers/pasoe/extras/keydist.war
```

For more information, see:

- Configure STS client key management using stskeyutil on page 98
- Configure STS client key management using the AdminServer STSKey plugin on page 99

Configure STS client key management using stskeyutil

OpenEdge administrators using STS client keys can use the STS client key management service offered in OpenEdge 12.3 for the OpenEdge Authentication Gateway server. STS client keys provide additional security measures, such as site isolation, to OpenEdge installations.

To configure remote STS client key management for OpenEdge installations using syskeyutil:

1. Deploy the Key Distribution application to the OpenEdge Authentication Gateway server that is configured with an STS server key, for example:

   ```
   proenv>oeauthserver/bin/tcman deploy %DLC%/servers/pasoe/extras/keydist.war
   ```

2. On each OpenEdge Installation that uses the OpenEdge Authentication Gateway server, run the `stskeyutil` command to generate a corresponding STS client key, for example:

   ```
   proenv>stskeyutil install -url https://hostname:port -keystorepath $DLC/keys
   [-nohostverify] -keydist -onlyifmodified
   ```

This command specifies the `-keydist` option, which indicates that Key Distribution is enabled. The stskeyutil remotely accesses the STS server key from the OpenEdge Authentication Gateway server at the requested URL (`-url`). The stskeyutil stores the encrypted STS client key in the client machine's `$DLC/keys` directory, as specified with the `-keystorepath` option. The `-onlyifmodified` option specifies that the OpenEdge client accesses the STS server key, and updates the STS client key, only if the STS server key is newer than the existing STS client key.

The optional `-nohostverify` switch should not be used in production, but can be used for development and testing. For more information about supported options for the `stskeyutil install` command, see stskeyutil install utility.

Before the STS client key management service was added to OpenEdge, administrators had to use the stskeyutil command to access a copy of the STS server key, which had to be hosted on each client's OpenEdge installation. Administrators had to issue an stskeyutil such as the following:

```
proenv>stskeyutil install -url https://hostname:port -keystorepath $DLC/keys -filename
 path-to-shared-drive/oests-server.ecp
```

The STS server key had to be on available to the client machine and specified using the `-filename` option. After the administrator generated the STS client key, the STS server key had to be manually removed from the client machine. This had to be done on every OpenEdge installation site.

With the STS client key management service, OpenEdge client machines do not need to copy the STS server key locally in order to generate an STS client key. The stskeyutil command generates the STS client key using information obtained from the OpenEdge Authentication Gateway server using the Key Distribution application.

Configure STS client key management using the AdminServer STSKey plugin

In OpenEdge 12.3, an optional STSKey plugin was added to the OpenEdge AdminServer. This plugin gives the OpenEdge AdminServer the ability to maintain the STS client key on an OpenEdge installation.

Note: This procedure requires an OpenEdge AdminServer on each OpenEdge Authentication Gateway client's OpenEdge installation. If your site does not meet this requirement, then you can manage STS client keys using the stskeyutil utility on each client OpenEdge installation. For more information, see Configure STS client key management using stskeyutil on page 98.

The configuration for the STSKey plugin is defined in the `AdminServerPlugins.properties` file, which is located in the `$DLC/properties` directory. Accompanying the STSKey plugin for the AdminServer is a supporting `stskeyplugin.properties` file (also in `$DLC/properties`), which is where administrators define connection properties for the STS client key generation.

The STSKey plugin manages the STS client key of the OpenEdge Authentication Gateway client's OpenEdge installation using the stskeyutil utility and its ability to use the OpenEdge Authentication Gateway Key Distribution service. The STSKey plugin includes additional functionality over the stskeyutil utility, including specifying the frequency of when the plugin checks the Key Distribution service to see whether a new STS server key was installed on the OpenEdge Authentication Gateway server, and supporting encoded passwords for STS client key generation.

To use the AdminServer STSKey plugin to manage client OpenEdge installation's STS client keys:

1. Ensure the Key Distribution application is deployed on your OpenEdge Authentication Gateway server that is configured with an STS server key, for example:

```
proenv>oeauthserver/bin/tcman deploy %DLC%/servers/pasoe/extras/keydist.war
```

2. On the OpenEdge Authentication Gateway client's OpenEdge installation, stop the AdminServer:

```
proenv>proadsv -stop
```

3. Edit the `$DLC/properties/stskeyplugin.properties` file on the client machine to specify the OpenEdge Authentication Gateway server as the `keydistURL`. For more information about the additional supported properties, see STSKey plugin for the AdminServer.
4. Start the AdminServer on the OpenEdge client machine:

```
proenv>proadsv -start
```

   The AdminServer uses the Key Distribution service to periodically check the OpenEdge Authentication Gateway server for an STS server key, and if the STS server key changed, then it generates an encoded STS client key into the directory specified by the `keystorepath` field in the `stskeyplugin.properties` file. Then the AdminServer updates the client OpenEdge installation's STS client key. The default location is `$DLC/keys/`*`encoded-client-key`*`.ecp`).
5. Verify the connection to the OpenEdge Authentication Gateway server using the `stsclientutil` command, for example:

```
proenv>stsclientutil -nohostverify -url https://hostname:port -cmd authenticate
-user test -password test
```

   For more information about `stsclientutil`, see 'STS client utility (stsclientutil)' in *OpenEdge Getting Started: OpenEdge Authentication Gateway Guide*.

STS client key management logs for the AdminServer

The client-side logging from the AdminServer is shown in the `$WRK/admserv.log` file.

When the AdminServer successfully generates a new STS client key, the following message is printed to the `admserv.log` file:

```
stskeyutil request successful : key updated
```

If the AdminServer checks the Key Distribution service and finds that the STS server key did not change, then the following message is printed to the `admserv.log` file:

```
server key not modified since last client key update
```

Logging verbosity is set using the `verbose` property in the `stskeyplugin.properties` file. For more information, see STSKey plugin for the AdminServer.

Advanced logging for the STSKey plugin can also be configured using the AdminServer Log Console in OpenEdge Management.

Updating the STS server key and STS client keys

If the stskeyutil generates a new STS server key (using, for example, the 'stskeyutil create' utility in *OpenEdge Getting Started: OpenEdge Authentication Gateway Guide*, with the `-overwrite` option) for the OpenEdge Authentication Gateway server, the OpenEdge client machines' STSKey plugins check the OpenEdge Authentication Gateway server to see if a new STS server key was installed (the frequency of these checks is determined by the `pinginterval` setting). If the OpenEdge Authentication Gateway server has a new STS server key, and if the `onlyifmodified` option is set to 1 on the client machines, then the STSKey plugin updates the STS client key on the client machines.

You can verify the STS server key update by looking at the *`hostname`*`.access.`*`{date}`*`.log` file for the OpenEdge Authentication Gateway server, and checking the timestamp to see that a new STS server key was generated and sent by the Key Distribution application to the client machines.

STS client key management logging

To verify that the OpenEdge Authentication Gateway STS client key management application is operational:

1. Use the `stskeyutil -keydist` utility from the OpenEdge Authentication Gateway client's OpenEdge installation.
2. Check the OpenEdge Authentication Gateway's access log (for example, `oeauthserver/logs/hostname.access.`*`{date}`*`.log`) for a successful HTTPS `GET` status of `200`.

4

Configure the OpenEdge Database to use the OpenEdge Authentication Gateway

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

The following topics describe the steps to configure an OpenEdge database to communicate with the STS:

- Configure your OpenEdge database on page 103
- SSO Token Exchange on page 105
- STS Client Key (for database server) on page 106
- Start OpenEdge database server on page 107

For details, see the following topics:

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

Configure your OpenEdge database

This topic 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` as described in Data export utility on page 119 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 93.

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 (stsurlutil).

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).

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 proutil command, see PROUTIL ENABLEAUTHGATEWAY qualifier on page 165.

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 166.

SSO Token Exchange

This topic 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 Configure domains on page 79. 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 bolded 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 93.

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
```

Note: Beginning in OpenEdge 12.3, STS client key management can be handled by the Key Distribution service. For more information, see About STS client key management on page 97.

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 want to connect Client-Server, you need to generate the client key on both client and server.

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 `-stslogginglevel`. For more information, see STS logging level (-stslogginglevel) on page 164.

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. For more information, see Security cache size (-secsize) on page 163.
- `-certstorepath`, `-keystorepath`, `-nohostverify`, and `-nosessionresue` are shared between TLS-enabled connections and communication with the Authentication Gateway STS server. For more information, see the Startup parameters on page 159 documentation entries.

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 120.

5

Configure applications to use the OpenEdge Authentication Gateway

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

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

- The STS AuthenticationProvider on page 109
- Configuration and Testing on page 111

For details, see the following topics:

- The STS AuthenticationProvider
- Configuration and Testing

The STS AuthenticationProvider

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

The STS AuthenticationProvider:

- Takes user credentials (i.e. user name and password) as input

- Connects to the Authentication Gateway via a secure HTTPS connection
- Sends user credentials to the Authentication Gateway for authentication

If the authentication is successful, the STS AuthenticationProvider:

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

Spring Security URL authorization:

- Performs Role-Based Authorization (RBA)
- Passes the client-principal token (if RBA is successful) to ABL applications

---

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

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

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

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

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

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

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

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

Configuration and Testing

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

- OpenEdge Authentication Gateway configuration on page 111
- Configure a web application on page 113
- Troubleshoot the PAS for OpenEdge instance on page 114

See OpenEdge Authentication Gateway utilities on page 118 for information about the command line tools that are available to configure and manage the Authentication Gateway.

OpenEdge Authentication Gateway configuration

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

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

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

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

Note: Beginning in OpenEdge 12.3, STS client key management can be handled by the Key Distribution service. For more information, see About STS client key management on page 97.

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

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

Note:

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

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

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

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

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

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

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

Test the connection

Once the HTTPS (TLS) certificates and client-key are installed, they should be tested to ensure that the PAS for OpenEdge instance can connect to the OpenEdge Authentication Gateway. To test the connection:

1. Test access to the installed Authentication Gateway client-key with the `stskeyutil verify`command. For example:

   ```
   oe_install_dir/bin/stskeyutil verify -file oeagserverkey.ecp -info
   ```

   Successful verification is indicated by the following message:

   ```
   Server key validation success
   ```

2. After verifying that the Authentication Gateway is running, test HTTP connectivity to the OEAG server (ensure the server is running before you do this)

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

Note: Only use `-nohostverify` when connecting to an Authentication Gateway that is using the OpenEdge-supplied default server certificate, which is used in application development or testing situations but not for actual application deployment.

See STS Key Utility (stskeyutil) on page 123 and STS client utility (stsclientutil) on page 120 for more information about these utilities and their command-line options.

Configure a web application

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6. (Optional) Use the sniHost property when the OpenEdge Authentication Gateway server is configured with multiple virtual hosts that are bound to a single IP address. Specify the hostname that you want your web application to connect to.

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

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

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

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

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

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

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

Troubleshoot the PAS for OpenEdge instance

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

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

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

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

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

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

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

6

Reference entries

This chapter contains reference entries for:

- OpenEdge Authentication Gateway utilities
- ABL reference
- Startup parameters
- Database utilities
- Glossary of Terms on page 167

For details, see the following topics:

- OpenEdge Authentication Gateway utilities
- ABL reference
- Startup parameters
- Database utilities
- Glossary of Terms
- STSKey plugin for the AdminServer

OpenEdge Authentication Gateway utilities

The following topics (and related subtopics) contain information about the command line utilities used for managing and configuring the OpenEdge Authentication Gateway:

- Change the keystore password utility (changeP12pwd) on page 118
- Data export utility on page 119
- STS client utility (stsclientutil) on page 120
- STS Key Utility (stskeyutil) on page 123
- STS Connection Role Utility (stsconnroleutil) on page 130
- STS URL Utility (stsurlutil) on page 137
- Strong Password Encryption Utility (stspwdutil) on page 140

Change the keystore password utility (changeP12pwd)

This utility changes the keystore password in a specified keystore file. The utility is found in the OpenEdge installation's `/bin` directory (`%DLC%\bin`).

The default password is `password` in the default keystore file: `%DLC%/bin/tomcat-keystore.p12`. This 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. This task is described below.

Syntax

```
changeP12pwd { -keystorefile file_name -keypassword current_password } | -h
```

`-keystorefile` *`file_name`*

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

`-keypassword` *`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>cd %WRKDIR%\oepas1\conf
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 utility used in troubleshooting OpenEdge database connections to the Secure Token Server (STS) application. Specifically, stsclientutil addresses these tasks:

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

Command syntax

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

-cmd *command*

Specifies the task to perform from one of the following:

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

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

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

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

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

  ```
  -domain domain
  ```

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

`-url` *sts-url*

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

command-options

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

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

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

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

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

`-installpath` *dlc-path*

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

`-keystorepath` *ks-path*

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

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

`-certstorepath` *`cs-path`*

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

`-logginglevel` *`level`*

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

`-nohostverify`

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

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

`-servername`

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

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

Examples

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

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

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

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

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

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

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

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

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

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

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 topics that follow:

- stskeyutil common options on page 123
- stskeyutil create utility on page 124
- stskeyutil verify utility on page 125
- stskeyutil install utility on page 125
- stskeyutil keyfile utility on page 127
- stskeyutil keytest utility on page 128
- stskeyutil headertest utility on page 129
- stskeyutil ssotest utility on page 129

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

The stskeyutil install command creates a new STS client key file from an STS server key file.

```
stskeyutil install -url sts-url
  [ -keystorepath path ]
  [ -file filename ]
  [ -overwrite ]
  [ -keydist ]
  [ -node node-name ]
  [ -dlcpath oeinstalldir ]
  [ -sslprotocol sslprotocol ]
```

```
   [ -sslciphers sslciphers ]
   [ -certstorepath certstorepath ]
   [ -servername servername ]
   [ -nohostverify ]
   [ -onlyifmodified ]
   [ -v ]
   [ -silent ]
```

`-url sts-url`

Specifies the STS server's URL in the format: `https://hostname[:port]/[web-app]`.

`-keystorepath path`

Specifies access to the STS server key file in this directory. The default is `$DLC/keys`.

`-file filename`

Specifies the STS server key filename (for example, `oests-key.ecp`) from the `-keystorepath` directory. This option is required when manually managing STS client keys. This option is not required when

`-overwrite`

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

`-node node-name`

Specifies the cluster node named *node-name*.

`-dlcpath oeinstalldir`

Path to OpenEdge installation directory. The default is `$DLC`.

`-sslprotocol sslprotocol`

Specifies the SSL protocol name. The default is `TLSv1.3`.

`-sslciphers sslciphers`

Specifies the SSL cipher.

`-certstorepath certstorepath`

Path to certificate store. The default is *$DLC/certs*.

`-servername servername`

Hostname that supports the server name indicator (SNI).

`-keydist`

In OpenEdge 12.3 and above, the `-keydist` option accesses the STS server key from the OpenEdge Authentication Gateway server specified in `-url`. The server must have the Key Distribution application installed to enable this functionality. For more information, see About STS client key management on page 97.

`-nohostverify`

Does not verify the hostname's server certificate. Do not use this option in production.

`-onlyifmodified`

Generates a new STS client key only if the STS server key is newer.

`-v`

Generates verbose output.

`-silent`

Utility does not prompt for password, access-codes, or filepaths.

About syskeyutil install

The `stskeyutil install` command 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. Having a separate STS client key 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 the setup 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 stskeyutil install command. To enable generating a STS client key for each cluster node, follow this process:

- Ensure that each cluster node uses the same absolute path to the OpenEdge installation.

  Warning: If this is not true, stop now!

- Obtain each cluster node's nodename (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.
- Execute `stskeyutil install` 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 topics 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.

Notes

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.

Notes

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
     -grantee 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 (stsurlutil)

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 topics that follow:

- stsurlutil list utility on page 137
- stsurlutil status utility on page 138
- stsurlutil update utility on page 138

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`*

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

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

`-servername` *`virtual-hostname`*

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

`-folder` *`folder-name`*

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

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

`-ssl` *`args`*

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

`-test true | false`

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

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

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

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

Specifies additional databases and connection options.

abl-startup-options

Specifies additional ABL connection parameters.

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

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
```

ABL reference

The topics that follow contain new and updated ABL constructs related to the OpenEdge Authentication Gateway.

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 151 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
    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 151 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 151

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.

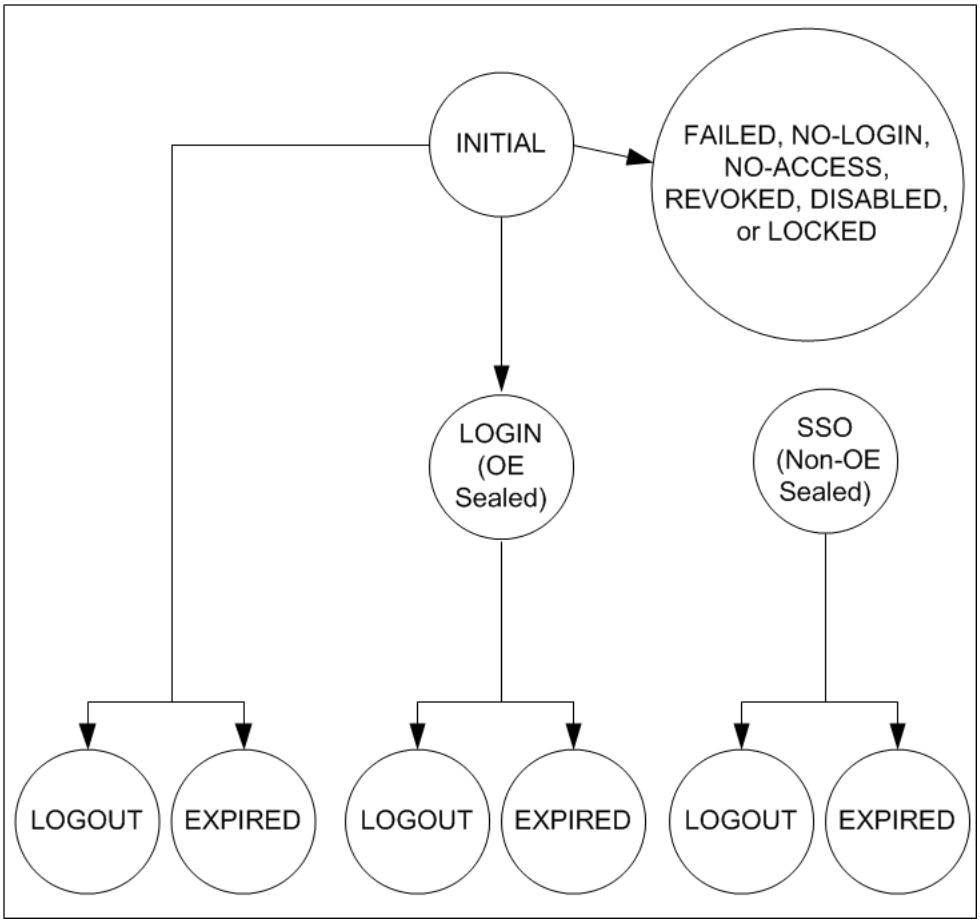
Table 1: Client-principal LOGIN-STATE attribute values

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 151, SECURITY-POLICY:SET-CLIENT( ) method, the SET-DB-CLIENT function, or via an OpenEdge STS.

This value...	Is set...
"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 153, 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.
"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 153 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 149 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 149, VALIDATE-SEAL( ) method (enhanced for STS) on page 153

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 156	AccessRevoked property on page 156	AccountDisabled property on page 156
AccountExpired property on page 156	AuthenticationAccess property on page 157	AuthenticationFailed property on page 157
Custom property on page 157	InvalidConfiguration property on page 157	LoginDenied property on page 157
LoginLockout property on page 158	MaxTries property on page 158	MissingCredentials property on page 158
PasswordExpired property on page 158	PermissionDenied property on page 158	Success property on page 159
UnknownUser property on page 159	–	–

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 *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 *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 *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 *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 *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 *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 *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 *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 *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 *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 *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 *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 *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 *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 *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 *ABL Reference*)

Startup parameters

The topic that follow contain new startup parameters related to the OpenEdge Authentication Gateway.

Certificate store path (-certstorepath)

Use Certificate store path (`-certstorepath`) to specify the directory location of the Certificate Authority (CA) certificates to validate an 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 162 for more information on how these parameters work together.

Keystore path (-keystorepath)

Use Keystore path (`-keystorepath`) to specify the directory location of an 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 keystore is located.

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

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

Use No Host Verify (`-nohostverify`) to turn off host verification for a Transport Layer Security (TLS) 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 TLS enabled for communication to its remote clients, the communication to the OpenEdge Authentication Gateway server is always run with TLS enabled.

If the database is running with TLS enabled, No Host Verify (`-nohostverify`) is set the same for both the OpenEdge Server TLS 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 Transport Layer Security (TLS) session ID to reconnect to the same TLS-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 TLS enabled for communication to its remote clients, the communication to the Authentication Gateway STS Server is always run with TLS enabled.

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

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

Note: OpenEdge TLS turns on TLS 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 TLS session and ignores any different connection parameters that are specified for the subsequent connection, including `-nosessionreuse`. If you want to change TLS 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 TLS 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 160 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 Configure domains on page 79 for more information.
- When `-OSUser` is used alone (without the Domain (-domain) (for STS capability) on page 160 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 160 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.
```

The Security Cache Size (`-secsize`) parameter may be modified while your database is online using PROUTIL INCREASETO.

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

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 or greater than 5 generate an error

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.

The STS logging level (`-stsloggінglevel`) parameter may be modified while your database is online using PROMON or through the _DbParams VST. This online change will only take effect on new connections to the database.

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.
- 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 topics that follow describe the database utilities that support the OpenEdge Authentication Gateway:

- PROUTIL ENABLEAUTHGATEWAY qualifier on page 165
- PROUTIL DISABLEAUTHGATEWAY qualifier on page 166
- Auditing events for OpenEdge Authentication Gateway on page 166

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 more information, see "STS URL Utility (stsurlutil)" 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.

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.

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

Event ID	Event Name	Event Type	Event Description	Policy Name
10612 **Note:** Connecting SQL via the Authentication Gateway is not supported in Release 11.6.2.	_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.
- *`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.

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

STSKey plugin for the AdminServer

The STSKey plugin for the AdminServer is defined in the `$DLC/properties/stskeyplugin.properties` file. In this file, you configure the stskeyutil settings for the AdminServer to access the OpenEdge Authentication Gateway server to retrieve the STS server key.

The following table provides information about the properties:

Property	Description
stskeyExecFile	The path to the stskeyutil executable. The default is $DLC/bin/stskeyutil.
keydistURL	The hostname and port of the OpenEdge Authentication Gateway server.
node	The cluster node name.
onlyifmodified	If set to 1, the client downloads the STS server key, and updates the STS client key, only if the STS server key is newer than the existing STS client key. The default is 1.
dlcpath	The path to the OpenEdge installation. The default is $DLC.
sslprotocol	The SSL protocol to use. The default is TLSv1.3.
sslciphers	The SSL cipher to use.
certstorepath	The path to the OpenEdge certificate store. The default is $DLC/certs.
keystorepath	The path to the OpenEdge keystore. The default is $DLC/keys.
nohostverify	The option to suppress hostname verification during the SSL/TLS handshake. The default is 0. Do not enable in a production environment.
verbose	The diagnostic logging level. Supported values are 0 through 4. The default is 0, which indicates that verbose logging is off.
serverkeypwd	The encoded password for the downloaded STS server key that is used to create the STS client key. This property also supports clear text passwords.
pinginterval	The frequency that the plugin checks with the Key Distribution service to see if a newer STS server key was installed. Specified in minutes. The default is 60.
servername	The SNI server name. The default is blank.