Progress®



Learn about Identity Management

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....7

What is Identity Management?....9

What is Identity?....10
How OpenEdge supports user account systems....10
Authentication and user account systems....10
User credentials authenticated by OpenEdge....12
Specifying a user ID for OpenEdge authentication....13
Common authentication mechanism for database clients....14
OpenEdge identity types and their applications....14
Authenticated user identity....14
Default user identity (for backward compatibility)....17
Authentication mechanisms for other OpenEdge components....19
User identity and multi-tenancy....19

How is OpenEdge Identity Managed?....21

OpenEdge authentication....21
Authentication operations....22
User authentication process....23
Single sign-on (SSO) process....24
OpenEdge authorization....25
Tenant authorization....25
Role-based authorization....26

Configuring and Implementing Authentication in OpenEdge....29

Defining and configuring security domains....30
Requirements to configure a domain....30
Defining the name of a domain....31
Defining and specifying the system type....31
Entering a domain access code....33
Enabling and disabling domains....34
Entering system options....34
Identifying the tenant to which a domain belongs....34
Pre-configured and reserved OpenEdge domains....35
Configuring authentication-enabled domains....35
Configuring authentication....36
Run-time domain configuration....37

Entering user credentials in OpenEdge....38
Authentication in ABL applications....38
Initializing a client-principal object for user authentication....39
Exporting and importing a client-principal object....39
OpenEdge-performed authentication and SSO....40
Application-performed user authentication....41
Managing identity for multi-tenancy....42

Configuring and Implementing Authorization in OpenEdge....51

OpenEdge authorization models....51
Non-multi-tenant vs. multi-tenant authorization....52
Tenant data access....52
When a user's domain is available for access control....53
User ID patterns as ACLs and ABL permissions checking....54
Patterns affecting ABL permissions....54

Glossary....59

Preface

Purpose

This manual defines identity management and describes the features for managing user identity in OpenEdge.

Audience

Any developer, database administrator, security administrator, software architect, or other person responsible for designing, implementing, or managing secure access to an OpenEdge installation.

Organization

- What is Identity Management? on page 9

  Provides an overview of both identity management and identity management in OpenEdge.

- How is OpenEdge Identity Managed? on page 21

  Provides an overview of the essential processes for managing OpenEdge identity.

- Configuring and Implementing Authentication in OpenEdge on page 29

  Describes the features for configuring and implementing authentication in OpenEdge.

- Configuring and Implementing Authorization in OpenEdge on page 51

  Describes the features for configuring and implementing authorization in OpenEdge.

- A glossary of terms related to identity management.

Documentation conventions

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

Purpose

This manual defines identity management and describes the features for managing user identity in OpenEdge.

Audience

Any developer, database administrator, security administrator, software architect, or other person responsible for designing, implementing, or managing secure access to an OpenEdge installation.

Organization

- What is Identity Management? on page 9

  Provides an overview of both identity management and identity management in OpenEdge.

- How is OpenEdge Identity Managed? on page 21

  Provides an overview of the essential processes for managing OpenEdge identity.

- Configuring and Implementing Authentication in OpenEdge on page 29

  Describes the features for configuring and implementing authentication in OpenEdge.

- Configuring and Implementing Authorization in OpenEdge on page 51

  Describes the features for configuring and implementing authorization in OpenEdge.

- A glossary of terms related to identity management.

Documentation conventions

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

1

What is Identity Management?

Identity management, in OpenEdge, is a set of systems and tools to control the user identities encountered by OpenEdge. This system provides a number of options to manage identity without the need for application code changes.

Note: The OpenEdge identity management features described in this manual apply primarily to the OpenEdge RDBMS and its immediate clients, including ABL clients, SQL clients, and database command-line utilities. Identity management for other OpenEdge features or external interfaces to OpenEdge, such as Web browsers, Web servers, and Open Clients have their own security services and are beyond the scope of this manual.

The purpose of identity management is to ensure that the resources of an information system—including applications and data—are accessed only by those users who are trusted to access them and are accessed in a manner that is appropriate for each individual user or particular group of users. Indeed, an essential function of identity management is to protect information resources so only authorized users can access the information system itself. It does this through an *authentication system* that serves as the gateway for all access to the information system. Once access is granted to a particular user, each attempt to access protected resources is controlled by an *authorization system* that determines if and how the user can access that resource. The security and capabilities of an identity management system are often part of a larger *security system* consisting of hardware and software that provides authentication, authorization, encryption, and auditing services to an information system.

OpenEdge provides its identity management system as a core business service of the product. This means that essential features of OpenEdge identity management are both supported by the product and shared in common by multiple OpenEdge components. This document provides an overview of that support and how to use it to secure access to OpenEdge components and the applications you build with them.

For details, see the following topics:

- What is Identity?

- How OpenEdge supports user account systems
- OpenEdge identity types and their applications
- User identity and multi-tenancy

What is Identity?

Identity (or *user identity*) is the means by which a user can be securely known to a software system and is typically represented in the software by a security token. A *security token* is an object that contains both user credentials and additional information about the user's roles and capabilities. *User credentials* (sometimes referred to as *login credentials*) consist of the information required to authenticate the user against a secure user account system known to the authentication system or application. The *user account system* manages a repository of user accounts and verifies that the login credentials asserted by the security token match valid account in the repository. A security token supports the authentication process by transporting a set of user credentials to be authenticated. Upon successful authentication, the security token becomes a read-only (*sealed*) container for transporting and asserting the authenticated user's identity to access authorized resources. Once it is sealed, the security token also represents a *login session*, which is a period of time during which the security token represents a valid user identity.

The security token information about roles and capabilities authorizes the user to access specific resources of the software system. This resource authorization is supported by *access control lists*, which grant or deny access to particular resources for authenticated user identities.

All assertions of identity in OpenEdge are represented by a security token instance, and OpenEdge exposes security token objects to an ABL application in a manner that supports flexibility in the design and implementation of its authentication model. For authentication processes managed by OpenEdge and its installed components, the security token is hidden.

How OpenEdge supports user account systems

OpenEdge provides built-in support for specific user account systems and supports two different mechanisms for ABL clients to authenticate user identities against user account systems that do not have built-in OpenEdge support. This support includes configurable authentication systems that manage access to specified user account systems.

Authentication and user account systems

Depending on the authentication system configuration, an asserted user identity can be validated through user authentication performed by:

- **OpenEdge** — In a built-in operation, OpenEdge accepts the user credentials (which identify a user and authentication system) and calls out to the authentication system to validate the specified user identity through a configured user account system. Depending on the success or failure of the user validation, OpenEdge then seals a security token that represents the user identity. For a successful user validation, OpenEdge also establishes the user identity in an OpenEdge security system, which might be an ABL session and one or more OpenEdge database connections, depending on the operation and OpenEdge resources involved.
- **An ABL application** — An ABL application accepts the user credentials and calls out to an ABL-implemented or external user account system to validate the user identity. Depending on the success or failure of the

user validation, the application typically executes an OpenEdge operation to seal a security token that represents the user identity. For a successful user validation, the application also typically executes another OpenEdge operation using the sealed security token to establish the user identity in an OpenEdge security system, which might be an ABL session and one or more OpenEdge database connections, depending on the operation and OpenEdge resources involved.

The result of a successful OpenEdge user authentication is always a sealed security token, which an ABL application can use in an OpenEdge single sign-on operation to establish the same authenticated user identity in additional ABL sessions and database connections. A *single sign-on* (SSO) operation validates that the authenticated user identity a sealed security token represents is acceptable to a given security system, and if successful, establishes that identity in the security system. If the ABL application performs the user authentication and seals a security token with the authenticated user identity, it can also use this security token in OpenEdge SSO operations to establish the user identity in additional ABL sessions or database connections. For more information on how OpenEdge manages user authentication and SSO operations, see OpenEdge authentication on page 21.

The following sections describe how OpenEdge supports user accounts in authentication systems. For more information on user accounts in OpenEdge, see OpenEdge identity types and their applications on page 14.

User accounts for OpenEdge-performed authentication

For supported user account systems, OpenEdge performs user authentication using two built-in authentication systems. These built-in authentication systems provide OpenEdge access to either an internal user account system built into the OpenEdge RDBMS or the local operating system (OS) user account system where OpenEdge is installed. Using one of these built-in authentication systems, ABL clients, SQL clients, and the command-line database utilities can all authenticate users defined in either the OpenEdge RDBMS user accounts or the local OS user accounts.

OpenEdge also allows you to configure user-defined authentication systems for use by ABL clients only that allow OpenEdge to perform the user authentication. These user-defined authentication systems are configured with ABL authentication callbacks that either implement their own user account systems or manage access to external user account systems, such as LDAP or OpenID. When an OpenEdge authentication operation uses one of these user-defined authentication systems, OpenEdge invokes the associated ABL callback to perform the user authentication. Thus, OpenEdge can authenticate user identities against external user accounts in the same way that it uses built-in authentication systems to authenticate against the database internal and OS user accounts.

In addition, OpenEdge allows you to configure these built-in authentication systems with ABL authentication callbacks that extend the built-in user authentication operations executed against the built-in user account systems. This allows you to add additional criteria, such as times of day or other login limits, to further validate an otherwise successful user authentication. You can also configure both built-in and user-defined authentication systems with ABL callbacks that perform customized processing after any OpenEdge user-authentication or SSO operation establishes a user identity in an ABL session or database connection.

User credentials authenticated by OpenEdge

The user credentials required for an OpenEdge authentication include:

- **User name** — A character string of varying length that specifies the user's account name (which can be blank) and whose exact format depends on the authentication system that stores and secures the user's account information. However in OpenEdge, the user name cannot contain the `'@'` character, because this character delimits the name of a security domain in a user ID (see following bullet and Specifying a user ID for OpenEdge authentication on page 13). In OpenEdge, a user is always a member of an OpenEdge security domain, with backward-compatible support for previous OpenEdge releases that do not explicitly define security domains.
- **Domain name** — A case-insensitive character string of varying length between zero (0) and 64 characters that specifies the name of an OpenEdge security domain of which the user is a member, with a format

similar to the domain of an E-mail address. In general, a *security domain* (or, simply, *domain*) includes an application, or collection of applications, whose security systems are configured to create and trust a common security token for authentication and authorization. In OpenEdge, a security domain is defined in an OpenEdge RDBMS and is configured with:

- A name (which can be blank, the default)
- A single authentication system used to authenticate the identity of all users who are members of the domain
- A secret access code used to cryptographically seal the security token after a successful user authentication, and also used to later validate the user identity represented by the sealed security token.
- An indication that the domain is enabled for use at run time

In a multi-tenant database environment, the domain also identifies the tenancy of all users whose identity is authenticated in the domain. For more information on user identity and multi-tenancy, see User identity and multi-tenancy on page 19. For more information on OpenEdge security domains and their configuration, see Defining and configuring security domains on page 30.

- **Secret Passphrase** — Also known as a user account *password*, a character string of varying length whose contents are known only to the owner of the user account and the user account system. This value must match the corresponding passphrase stored by the user account system together with the user account information. The user account system might be configured to require certain characters in the value, for example, at least one upper-case letter, one numeral, and one special character from a given set.

After a successful OpenEdge-performed authentication, OpenEdge seals the user and domain name in a security token. If an ABL application authenticates the identity, regardless of the form of user credentials that it verifies, the result of successful authentication must also include a user and domain name that the application seals in a security token in order to manage the authenticated identity with OpenEdge identity management features.

User accounts for ABL application-performed authentication

An ABL application can implement the user authentication for a user-defined authentication system that is **not** configured with an ABL authentication callback. In this case, the application performs the user authentication directly against a user account system that it either implements itself or accesses externally, similar to an ABL authentication callback. However, an application-performed authentication can support a larger variety of user credentials, depending on the user account system. For example, the user credentials might consist of a biometric signature, such as a finger print, if that is what the user account system requires. However, if it is to work with other OpenEdge identity management features, the result of any successful application-performed user authentication must include the same user credentials to seal in a security token that are required for any OpenEdge-performed user authentication. For example, this allows the application-authenticated user identity to be validated and established in the ABL session or available database connection using an OpenEdge SSO operation.

User credentials authenticated by OpenEdge

The user credentials required for an OpenEdge authentication include:

- **User name** — A character string of varying length that specifies the user's account name (which can be blank) and whose exact format depends on the authentication system that stores and secures the user's account information. However in OpenEdge, the user name cannot contain the `'@'` character, because this character delimits the name of a security domain in a user ID (see following bullet and Specifying a user ID for OpenEdge authentication on page 13). In OpenEdge, a user is always a member of an OpenEdge

security domain, with backward-compatible support for previous OpenEdge releases that do not explicitly define security domains.

- **Domain name** — A case-insensitive character string of varying length between zero (0) and 64 characters that specifies the name of an OpenEdge security domain of which the user is a member, with a format similar to the domain of an E-mail address. In general, a *security domain* (or, simply, *domain*) includes an application, or collection of applications, whose security systems are configured to create and trust a common security token for authentication and authorization. In OpenEdge, a security domain is defined in an OpenEdge RDBMS and is configured with:

  - A name (which can be blank, the default)
  - A single authentication system used to authenticate the identity of all users who are members of the domain
  - A secret access code used to cryptographically seal the security token after a successful user authentication, and also used to later validate the user identity represented by the sealed security token.
  - An indication that the domain is enabled for use at run time

  In a multi-tenant database environment, the domain also identifies the tenancy of all users whose identity is authenticated in the domain. For more information on user identity and multi-tenancy, see User identity and multi-tenancy on page 19. For more information on OpenEdge security domains and their configuration, see Defining and configuring security domains on page 30.

- **Secret Passphrase** — Also known as a user account *password*, a character string of varying length whose contents are known only to the owner of the user account and the user account system. This value must match the corresponding passphrase stored by the user account system together with the user account information. The user account system might be configured to require certain characters in the value, for example, at least one upper-case letter, one numeral, and one special character from a given set.

After a successful OpenEdge-performed authentication, OpenEdge seals the user and domain name in a security token. If an ABL application authenticates the identity, regardless of the form of user credentials that it verifies, the result of successful authentication must also include a user and domain name that the application seals in a security token in order to manage the authenticated identity with OpenEdge identity management features.

Specifying a user ID for OpenEdge authentication

A *user ID* is a character string that contains the public representation of a user's identity. When a user provides their user credentials to access a protected resource, OpenEdge allows, and in some cases requires, the user and domain to be specified as a single character value, with the domain name delimited by the `'@'` character, using this basic syntax:

Syntax

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

If a user ID is specified without a domain, OpenEdge assumes the default domain. This format, which is similar to an E-mail address, is a *fully qualifieduser ID*. In certain cases, OpenEdge allows the user to specify only *`user-name`* (without a domain name) as a user ID, which is referred to as a *non-qualifieduser ID*, or simply the *user account name*. When a user specifies a non-qualified user ID for authentication, OpenEdge uses the blank domain as the user's domain, by default.

OpenEdge supports flexible options for entering user credentials, depending on the application or OpenEdge context, and sometimes prompts for the user and domain name as separate values. For more information on how an OpenEdge user can enter user credentials, see Entering user credentials in OpenEdge on page 38.

Common authentication mechanism for database clients

With a common security token to encapsulate user identity, OpenEdge employs the same basic user authentication mechanism for all OpenEdge database clients, including:

- Database connections for ABL clients, SQL clients, and database utilities
- Individual ABL sessions of all types
- ABL applications of all configurations, including client-server, multi-tier, service-oriented (SOA), and software as a service (SaaS)

The available authentication features, however, depend on the OpenEdge components involved, the configuration of their security domains, and in the case of ABL sessions and applications, the design of the application authentication model. Authorization support also differs, with one authorization system for SQL clients and another for all other OpenEdge components.

OpenEdge identity types and their applications

OpenEdge supports two basic types of user identity for OpenEdge database clients that can be applied in various ways:

- Authenticated user identity on page 14 — All applications
- Default user identity (for backward compatibility) on page 17 — Database connections only

In addition, OpenEdge supports different mechanisms to authenticate identities for OpenEdge components other than OpenEdge database clients. For more information, see Authentication mechanisms for other OpenEdge components on page 19.

Authenticated user identity

An authenticated user identity is any user identity that is authenticated using supported authentication systems as follows:

- Authentication systems that OpenEdge user authentication operations can use to directly perform the authentication:
  - **Built-in `_oeusertable`** - This authentication system is configured with the OpenEdge internal user account system defined in the hidden `_User` table of OpenEdge databases. This user account system is managed entirely by OpenEdge database administration tools. Users defined in the database `_User` table accounts are defined with both a user name and the name of an OpenEdge domain, which is always configured with the `_oeusertable` authentication system. The `_User` table accounts also support a blank user identity in which both the user name and domain are defined with the empty string (""), and the OpenEdge blank domain serves as the default domain for any user ID specified with only a user name.
  - **Built-in `_oelocal`** - This authentication system is configured with the native user accounts on the Windows or UNIX-based operating system where OpenEdge runs, with all of the user account security that the particular operating system provides. Users defined in Windows user accounts can be identified

by both a user name and a Windows domain name. UNIX user accounts support only user names within a single domain. However, authentication to any local OS user account requires a user ID that includes the OS-defined user name and the name of an OpenEdge domain that is configured with the `_oslocal` authentication system. OpenEdge domains defined to work on a Windows system can also be mapped to particular Windows domains, as required.

- **User-defined authentication systems** - These authentication systems are configured with ABL authentication callbacks that manage user accounts or access external user account systems for user authentication. These user-defined authentication systems must also be enabled for user authentication. To work with OpenEdge user authentication operations, owners of the user accounts accessed by an ABL authentication callback must be defined with a user name in an OpenEdge domain that is configured with a user-defined authentication system that is, itself, configured with the same ABL authentication callback. For more information on user-defined authentication systems and configuring them with ABL authentication callbacks, see Defining and specifying the system type on page 31.

  Caution: If you implement a user-defined authentication system using an ABL authentication callback Progress Software recommends that you run the authentication system only in a locked down server environment where the r-code for the callback can be secured.

Note: The OpenEdge blank domain can be configured with any authentication system, including `_oslocal` (which you might want to do on UNIX, for example), but only if no users in the database `_User` table accounts are defined with the blank domain.

- Authentication systems that require an ABL application to implement and perform the user authentication using application-managed user accounts or an external user account system, similar to ABL authentication callbacks. These authentication systems include the OpenEdge built-in `_extsso` authentication system and any user-defined authentication system that you enable for OpenEdge SSO only. An application-implemented authentication system is enabled for SSO only so any OpenEdge domain that is configured with that authentication system cannot be part of a user identity that OpenEdge can authenticate, but once authenticated (by the application) OpenEdge can establish using an SSO operation. An SSO-only authentication system can be configured with an ABL callback to perform post-processing after an OpenEdge SSO operation establishes a user identity that is authenticated in a corresponding domain.

  Caution:

  Securely implementing an authentication system requires extensive expertise with security systems. This manual provides some guidance on how to configure OpenEdge to maximize the security of an application-implemented authentication system. However, this information provides only a basis for implementing an authentication system in ABL and is by no means sufficient or complete.

  Progress Software recommends that you run an application-implemented authentication system only in a locked down server environment where the r-code for the application can be secured.

The following table provides an overview how an authenticated OpenEdge user identity can be applied.

Table 1: Authenticated OpenEdge user identity applications

Identity application	Description
Database connection identity	A user identity that has been authenticated for or assigned to the database connection. An OpenEdge RDBMS authorizes all access to specific database tables and fields using the database connection identity. The identity for a database connection can be authenticated directly by OpenEdge when the connection is established from an ABL client, SQL client, or database utility. In ABL, the initial database connection identity can be bound to the identity already established for the ABL session to which the database is connected (see ABL session identity in this table). Also in ABL, the initial database connection identity can be changed by authenticating a new identity or by validating a new identity from a previously sealed security token. **Note:** The ABL `USERID` function returns the current database connection ID (qualified or non-qualified, as appropriate) for a database connection, regardless of how it is set.
ABL session identity	A user identity that is associated with an ABL session, independent of any databases connected to the session. The ABL session identity can be used to authorize or identify user access to application features in a database-independent fashion. These can be features that are entirely application-defined or that are supported specifically by OpenEdge, such as the auditing identity. The identity for an ABL session can be authenticated directly by OpenEdge at the request of the ABL application, or the ABL application can perform its own authentication of the session identity as a user-defined authentication system. An ABL session can also set its identity by validating a previously sealed security token. For example, it can set its identity from the sealed security token representing the identity of an existing database connection or another ABL session (see ABL application identity). All options for authenticating or setting ABL session identity depend on ABL functions and methods that authenticate the identity and seal the security token for the session, or validate the session identity from a previously sealed security token.

Identity application	Description
ABL application identity	A common user identity established by a multi-tier application for use by all OpenEdge sessions that participate in handling a single user action or request. The is the same as the ABL session identity with its scope extended by application code to multiple ABL sessions and database connections. Typically, the application user identity is shared between a single Progress Application Server (PAS) for OpenEdge client and the PAS for OpenEdge multi-session agent that processes client requests. Depending on the application session model, this single application user identity can also be shared between a single ABL client session and multiple PAS for OpenEdge instances. In addition, any given ABL session can use this single application identity as both the ABL session identity and the identity of any or all database connections required by the session or application. You can set the application identity from a user ID that is authenticated in a single controlling session, which is typically a PAS for OpenEdge session that authenticates and manages the application identity for all ABL clients that log users into the application. For more information on multi-tier applications and application session models, see *OpenEdge Getting Started: Application and Integration Services*.
Auditing identity	The designated user identity that OpenEdge auditing stores in audit event records for an audit trail. There is no functionally independent auditing identity. Instead, the auditing identity is set from one of the other established identities, depending on the application configuration. The auditing identity for the audit trail recorded by any database is the current connection identity for that database, and this is the only source of auditing identity for SQL or database utilities. However, you can also set a database option to set the auditing identity from the identity of any ABL session that connects to the database, effectively making the database connection identity the same as the ABL session identity (see database connection identity in this table). In this way, you can configure auditing for every database that is connected from a given ABL session so that all audit trails for that session (or even the entire application) are associated with the same user identity. Also, any time you explicitly authenticate or set a new connection identity for a database, this becomes the new auditing identity for that database as well. **Note:** The ABL `USERID` function returns the user ID (qualified or non-qualified, as appropriate) for the current auditing identity, regardless of how it is set.

Default user identity (for backward compatibility)

OpenEdge supports a default user identity that does **not** require authentication for database connections from ABL or the database utilities. OpenEdge does not normally support a default identity for SQL, but again, for backward compatibility, does allow OpenEdge SQL to be configured to accept a form of default identity.

ABL default identity constraints

ABL supports a default user identity that is set when connecting to a database without providing a user ID and password. This default identity is applied in all situations where a database is connected, but the ABL startup parameters for specifying a user ID and password (`-U` and `-P`) are not used. OpenEdge tools written in ABL that prompt for a user ID and password (such as the character mode Data Dictionary or the Data Administration utility), apply the default user identity if the user cancels out of the prompt.

The user ID assigned for the default user identity depends on the domain configuration in the database. If at least one domain is enabled for run-time access that supports OpenEdge-performed user authentication, the default user identity is the blank user ID (blank user and domain name). If there is no such domain enabled, OpenEdge attempts to use the user's operating system (OS) user ID, but defaults to the blank user ID if an OS user ID cannot be resolved.

Note: OpenEdge assigns any default OS user ID the OpenEdge domain `"WINDOWSID"` or `"UNIXID"`, depending on the operating system.

If the default user ID is a valid OS user ID, the database can be configured to uniquely authorize access for it like any other authenticated user identity. However, if the default user ID is the blank user ID, OpenEdge **cannot** distinguish this default, unauthenticated user identity from an **authenticated** blank user ID. As a result, it is possible for a blank user identity to access the database without authentication.

Note: The blank user ID can be defined only in the database `_User` table accounts.

Caution: While often helpful to developers in a development environment, if you can, avoid relying on the blank user identity for any purpose in a deployment environment.

To prevent an unauthorized user from gaining access to an OpenEdge database using the blank default identity, OpenEdge supports database and security options that limit access using the blank user ID. You can prevent a blank user ID from connecting to a database to begin with. You can also prevent a blank user ID that connects to a database from gaining access to any tables and fields in that database.

Note: These settings block connection and data access to an **authenticated** blank user identity as well as to the **default** blank user identity.

Because, a database connection can have an authenticated blank user identity, the auditing identity can also be recorded as the blank user ID, which, again, cannot be distinguished from an unauthenticated, default blank user ID. So, if your database requires auditing, Progress Software strongly recommends that you prevent all access to the blank user ID.

Note: OpenEdge prevents any SSO operation from establishing an existing default user identity in any ABL session or any other database connection.

SQL default identity constraints

The SQL standard does not allow for a default user identity and requires the user to provide a valid user ID and password. However, if there are no domains enabled that are configured with a working authentication system, OpenEdge SQL does accept any non-blank user ID (ignoring the password) that the user enters, with the exception of the special DBA account, SYSPROGRESS. However, such a configuration is not typical for deployment.

Database utility default identity constraints

When starting up a database utility, such as PROUTIL or PROMON, if the user does not provide a user ID and password, the utility defaults to the operating system process identity. Therefore, at a minimum, database access from database utilities is authorized based on the user's operating system identity.

Authentication mechanisms for other OpenEdge components

OpenEdge uses different authentication mechanisms for the following components:

- **OpenEdge Management:**
  - **Database Administration Console (DAC)**—Authenticates user identities against an operating system file of user accounts on disk. Supports the option to use the DAC user ID and password for managed database connections.
  - **Unified Broker command-line utilities**—Authenticates user identities directly against he OS user accounts.
- **OpenEdge DataServers:**
  - **OpenEdge schema holder database**— uthenticates user identities in OpenEdge domains like any other OpenEdge database connection.
  - **Foreign data source**—Authenticates user identities as defined by the security system for the foreign data source. Some OpenEdge DataServers pass specified User ID (`-U`) and Password (`-P`) connection parameters to the foreign data source for authentication.

User identity and multi-tenancy

In a multi-tenant database environment, every user identity (including the blank default identity) has *tenancy*, which is the tenant or tenants whose data the user is authorized to access in the database. A user's tenancy is identified by the name of the single tenant that is configured for the user's domain, and the user's tenancy takes effect the moment their identity is authenticated and established in a multi-tenant database connection.

Note: The default (blank) domain is always configured for the default tenant. This configuration cannot be changed.

Because multiple users can be authenticated in the same domain, all of these users share the same tenancy. In addition, multiple domains can be configured with (“can belong to”) the same tenant. All users authenticated to any of these domains share the same tenancy and access the same tenant (and shared) data.

So, for each tenant defined in a database (including the default tenant), the database must define at least one domain configured with the tenant in order for users to access the database with the specified tenancy. Note that a given domain can be configured with only one tenant at a time.

Also, because every multi-tenant database defines its own set of tenants, a given domain defined in one database can be configured with a different tenancy than a domain of the same name defined in another database. So, for example, a given user identity might access one database as the regular tenant, `"General"`, while the same user identity might access another database as the super tenant, `"Master"`.

For an overview of multi-tenancy and access to multi-tenant databases, see *OpenEdge Getting Started: Multi-tenancy Overview.* For an overview of configuring domains for multi-tenancy, see Identifying the tenant to which a domain belongs on page 34.

2

How is OpenEdge Identity Managed?

The basic mechanisms for managing identity include user accounts, authentication, authorization, and the configuration of these security functions. The following sections provide an overview of how authentication and authorization is supported in OpenEdge for OpenEdge database clients, and where to find more information.

For details, see the following topics:

- OpenEdge authentication
- OpenEdge authorization

OpenEdge authentication

As described elsewhere (see *Learn about Security and Auditing)*, in OpenEdge, *authentication* is the process of:

- Ensuring that a user who accesses a software system is who they claim to be based on user credentials that they provide
- Providing a secure and portable means of identification (a sealed security token) for a successfully authenticated user, so their identity can be recognized throughout the system
- Validating this user identity in every component of the system before it can be used to authorize access to system features and data

OpenEdge provides a common authentication mechanism for all OpenEdge database clients, and the following topics describe the supported authentication operations. For a more detailed overview of how to configure and implement authentication, see Configuring and Implementing Authentication in OpenEdge on page 29.

Authentication operations

OpenEdge supports two basic types of authentication operations to authorize access to OpenEdge resources:

- User authentication — Authenticates user supplied credentials in a login operation against secure user accounts that are associated with a specific security domain. If successful, this operation produces a sealed security token that represents the user's identity in that domain. It then establishes (*sets*) the user identity in the security system where the user's identity is authenticated so the security system can authorize access to system features and data. Depending on the OpenEdge configuration, and the security systems involved, a single user authentication can set the same identity for more than one security system in a single operation.

  Also, as noted previously (see What is Identity Management? on page 9), OpenEdge supports two different mechanisms to perform user authentication, depending on how the user's security domain is configured:

  - **OpenEdge performed** — Domains configured for user authentication to the OpenEdge `_User` table (database `_User` table accounts) or to the operating system user accounts
  - **ABL Application performed** — Domains configured for authentication to any other (external) user accounts

- Single sign-on (SSO) — In ABL only, validates that the user identity has successfully passed authentication and the security system is configured to accept sealed security tokens before establishing the identity. Thus, an SSO operation can take a security token that has been created and sealed in one security system, and set the same user identity for a different security system within the same domain without requiring the user to login a second time. Depending on the OpenEdge configuration, and the security systems involved, a single SSO can set the same identity for more than one security system in a single operation.

OpenEdge security systems and authentication

OpenEdge security systems that these common authentication operations support include various types of database connections and ABL sessions. The types of database connections include connections from:

- ABL clients
- SQL clients
- Database command-line utilities, such as PROUTIL or PROMON

Note: OpenEdge predefines certain built-in security domains for use by database utilities only.

The types of ABL sessions include:

- Full ABL clients, including batch clients
- WebClients
- PAS for OpenEdge multi-session agents

Authentication operations supported for OpenEdge security systems

Of these security systems, all of them support access through user authentication. For backward compatibility, some support a form of default identity that does not require any authentication (see Default user identity (for backward compatibility) on page 17. Only ABL sessions, and database connections established from ABL sessions, support SSO operations. In addition, the ability to set the same user identity for more than one security system in a single authentication operation is supported only for ABL sessions and database connections. An ABL session can also retrieve the sealed security token from OpenEdge that was used to set the current identity for the session or any database connection established in the session.

User authentication process

All user authentication operations follow a common process and depend on a registry of enabled domains (domain registry) that has either been created in an ABL session or for each database connection. For more information on domains and domain registries, see Run-time domain configuration on page 37.

The following procedure applies to any OpenEdge-performed user authentication and many implementations of an application-performed user authentication. However, the form of user credentials required for an application-performed user authentication can be different from what is used to initialize the security token, requiring a different order to these steps. For more information, see Application-performed user authentication on page 41.

Process for a user authentication operation

A user authentication operation generally follows these steps to authenticate a user's identity:

1. If an unsealed security token is provided without the necessary user account claims, provides a new security token initialized with the user's credentials, including their user name, domain name, passphrase, and a unique login session ID.
2. If there is a login expiration time stamp set in the security token and it has expired, seals the security token in a state that indicates its time for authentication has expired and exits from the authentication operation with no further action.
3. Verifies that the name of the user credential's domain is found and enabled in the domain registry of the ABL session or database connection.
4. Verifies that the domain is configured with an authentication system that is enabled for user authentication.
5. Authenticates the user credentials against the user accounts defined for the domain's authentication system:
   a) Validates that an account exists for the user name and that its passphrase matches the passphrase in the user credentials.
   b) Validates that the user ID account is in an enabled state and passes all account conditions.
   c) If the authentication is successful:
      - Deletes the passphrase from the user credentials and erases all trace of it.
      - Loads the name of the authentication system and other documentary information about the authentication into the security token.
      - Seals the security token using the domain's access code.
      - Passes the authenticated identity to the authorization system for future use.

      If the authentication is **not** successful, it seals the security token in a state that indicates the authentication failed, and leaves the existing user identity for the ABL session or database connection unchanged.

Single sign-on (SSO) process

All SSO operations follow a common procedure and depend on a registry of enabled domains (domain registry) that has either been created in an ABL session or for a single connected database. For more information on domains and domain registries, see Run-time domain configuration on page 37.

Process for an SSO operation

An SSO operation generally follows these steps to validate and set the user identity using a sealed security token:

1. Retrieves the domain name from the security token and verifies that the domain exists in the domain registry of the ABL session or connected database, and is in an enabled state.
2. If there is a login expiration time stamp set for the security token, verifies that it has not expired, and if it has expired, sets the security token to a state that indicates its login time has expired and exits from the SSO operation without further action.
3. Performs a data-integrity check of the user credentials stored in the security token.
4. Validates the access code configured for the registered domain against the security token's seal.
5. If the tests in all of the previous steps are satisfied, the SSO operation is successful and continues with one of the following procedures, depending on the context:

   - Process for a successful SSO to an ABL session on page 24
   - Process for a successful SSO to an OpenEdge database connection on page 24

   If the SSO is **not** successful, the application continues with no effect on any existing user identity setting.

Process for a successful SSO to an ABL session

If the SSO operation is successful for an ABL session, it:

1. Makes a deep copy of the user credentials from the security token into the ABL session context.
2. Records the session's user ID and login session ID for internal use by auditing (and other internal functions), effectively setting the identity for the session.
3. Repeatedly executes the SSO operation in an attempt to set the same identity in each OpenEdge database connection currently established in the session, (see Process for a successful SSO to an OpenEdge database connection on page 1).

Note: In an ABL session, a database connection can be programmed to lock out the ability to set its identity when setting the session identity.

Process for a successful SSO to an OpenEdge database connection

If the SSO operation is successful for an OpenEdge database connection, it:

1. Makes a deep copy of the user credentials from the security token into the database connection context.
2. Sets the connection identity to the user identity represented by the security token.
3. If this is a multi-tenant database connection:

a) If the security token already stores the name of the database (from a previous connection), the corresponding tenant name and database tenant ID are assigned to the connection to authorize the user's tenancy.

b) If the security token does **not** already store the name of the database, the tenant name and database tenant ID are returned from the database domain configuration and the security token is resealed with the updated user credentials; then the tenant name and database tenant ID are assigned to the connection to authorize the user's tenancy.

For more information on tenant authorization, see Tenant authorization on page 25.

4. Audits the change in database connection identity.
5. For an ABL database connection, rebuilds the table and field permissions for the new user identity to use in any subsequent run-time or dynamic buffer access.

Note: For more information on ABL table and field permissions, see OpenEdge authorization on page 25

OpenEdge authorization

Authorization is the process of using a user's security token to control access to resources and the operations they can perform on those resources. These operations, including access to data, are typically controlled by a set of access controls associated with the user's identity. These *access controls* implement the authorization model used to regulate user access.

The OpenEdge RDBMS supports two different authorization models:

- **SQL**—A default closed model that prevents all access to databases, tables, fields, and views unless a specified access *privilege* is explicitly granted to a user. OpenEdge SQL clients follow this model.
- **ABL**—An default open model that grants all access to databases, tables, and fields unless an access *permission* is explicitly restricted from a user. All ABL sessions and database utilities follow this model.

For both models, the access privileges or permissions are set based on the user ID stored in the user's security token (see Specifying a user ID for OpenEdge authentication on page 13). For a non-multi-tenant database, only a non-qualified user ID (user name only) is recognized. For a multi-tenant database, a fully-qualified user ID (including the user and domain name) is both recognized and required. In the SQL model, to authorize privileges for a given user, you must specify the user's complete user ID, or a list of complete user IDs for more than one user. In the ABL model, to authorize permissions, you can specify lists of user ID patterns that can include wild-card and other characters that identify the access permissions for one or more users.

The following topics provide an overview of OpenEdge authorization components that are common to both of its authorization models, including tenant and role authorization. For a more detailed overview of how to configure and implement authorization for both models, see Configuring and Implementing Authentication in OpenEdge on page 29.

Tenant authorization

As explained in User identity and multi-tenancy on page 19, tenancy is an attribute of a user identity. So, every database connection identity has a tenancy in a multi-tenant database. This tenancy is established at the time the connection identity is established. Tenancy authorizes a user identity to access the data owned by, or accessible to, the tenant configured for their domain in a connected multi-tenant database. Note that a user's domain maps to the name of a tenant organization regardless of the tenant ID in each multi-tenant database.

Tenancy data models

All data in a multi-tenant database is allocated according to the following tenant access protocol:

- **Shared (non-tenant) data** — Accessible to all users, regardless of tenancy
- **Default tenant data** — Accessible to default-tenant users and to super-tenant users with the default effective tenancy (the OpenEdge default setting for super tenant effective tenancy)
- **Regular tenant data** — Accessible to regular-tenant users with the same regular tenancy and to super-tenant users with the same effective regular tenancy.

Regardless of tenancy, a user can only access data according to the SQL or ABL authorization settings for table and fields that have been established for their user ID in the database.

Managing tenancy access across multiple ABL sessions

When a user's identity is first established for a multi-tenant database connection, the user's tenancy is set using the tenant information stored in the database's own domain configuration. Because a domain in one database can be configured with a different tenant identifier than the same domain configured in another database, the same user identity can have a different tenancy in each multi-tenant database that it accesses.

To help manage the tenancy of a given identity used to access the same set of multi-tenant databases in different ABL sessions (such as a multi-session agent), ABL caches its tenancy in the security token for the identity whenever it is first used to authenticate a connection to a multi-tenant database. If the same sealed security token is then used to authenticate a connection to the same database in a different ABL session, the tenancy for the connection identity can then be authorized without having to retrieve the same tenant information from the database. Also, at any point, the application can query the security token for its tenancy in any multi-tenant database it has been used to authenticate a connection identity. For more information on ABL management of tenant authorization across databases and sessions, see Managing identity for multi-tenancy on page 42.

Role-based authorization

Both SQL and ABL authorization models support roles to which users can be assigned. When a user is assigned to a role, they can perform certain types of operations on and in the database that are permitted for their assigned role.

The basic roles include:

- **Database administrator** — Allows access to all database operations as any role initially, including the authorization to assign users to all other roles. Once users are assigned to other roles, the database administrator might no longer have authorization to perform the assigned roles. In the SQL model, there are a number of roles that a database administrator (DBA) always has the power to grant or revoke. In the ABL model, everyone is a database administrator. For more information on the database administrator role in the:
  - **SQL model** — See *OpenEdge Data Management: SQL Development*
  - **ABL model** — See *OpenEdge Data Management: Database Administration*
- Security administrator — Allows access to all database security operations, such as setting various database security options, configuring domains, defining users in the database `_User` table accounts, and assigning data access privileges and permissions to users. In the SQL model, the DBA is also the security administrator. In the ABL model, after a database administrator assigns the first security administrators, only the assigned

security administrators can act as security administrators and assign other security administrators. The default ABL security administrator is everybody and public.

Caution: If the blank user is the only ABL security administrator, do not add any non-blank users to the database `_User` table accounts. Once a non-blank user is added to the database `_User` table accounts, OpenEdge no longer allows a blank user to act as a security administrator, and all access to database security functions, if not to the database itself, is effectively locked out.

For more information on security administrators, see *OpenEdge Getting Started: Core Business Services - Security and Auditing* and *OpenEdge Data Management: Database Administration.*

- Audit roles — Allow access to various roles that have different audit privileges, including the audit administrator, application audit event inserter, audit data archiver, and audit data reporter. The assignment of audit roles follows a grant model. SQL DBAs and ABL security administrators are all audit administrators until they explicitly assign the first audit administrator. The audit administrator can assign users, including themselves, to all other audit roles. If only one user is assigned as an audit administrator, the database administrator can revoke that assignment and become the default audit administrator, again. Although the mechanism for assigning audit roles and privileges is different in SQL and ABL, settings made in either authorization model are recognized by the other, because auditing is a core service in common to both models. For more information on OpenEdge auditing and audit roles, see *OpenEdge Getting Started: Core Business Services - Security and Auditing.*

3

Configuring and Implementing Authentication in OpenEdge

The first task in setting up authentication for OpenEdge is to define and configure appropriate security domains. As noted previously, for OpenEdge, every user identity is defined in a domain. So, to authenticate users in a domain, OpenEdge must first have a configuration for that domain.

Users must be registered in the user account system supported by the domain. Note that if a user account system supports domains, they have no automatic relationship to OpenEdge domains. OpenEdge domains simply direct the authentication of user IDs to the appropriate authentication system.

At this point, SQL clients and database utilities are ready to accept logins. For an ABL application, the requirements might be more complex, especially for an multi-tier application with many moving parts. There are many authentication models that can be devised for a complex application configuration. So, designing and implementing one can require some care.

The following sections describe the OpenEdge features that support the entire process.

For details, see the following topics:

- Defining and configuring security domains
- Configuring authentication
- Run-time domain configuration
- Entering user credentials in OpenEdge
- Authentication in ABL applications

Defining and configuring security domains

You typically configure OpenEdge-supported domains in an OpenEdge RDBMS using database administration tools, including the:

- The Database Administration Console in OpenEdge Management
- Data Administration utility in the OpenEdge program group on Windows
- **Admin** menu of the character-mode Data Dictionary

OpenEdge also provides support for configuring domains in SQL and in ABL using an OpenEdge-installed ABL API. Each of these approaches to configuring OpenEdge domains supports different capabilities, which vary in usefulness and the level of security.

In general, an OpenEdge domain is identified by a name that is unique across the application for all domains. All domains are configured with the following settings:

- **A system type (authentication system)**—The type of authentication system that the domain relies on for authenticating the security token for all users in that domain. OpenEdge supports several built-in authentication system types and allows you to define new ones, including ones that allow you to define your own authentication system. Each type of authentication system supports a particular set of user accounts and specific operations to validate and set a given user identity.
- **An access code**—A secret value used by OpenEdge or your ABL application both to cryptographically seal a security token during user authentication and to validate a user's sealed security token, for example, to assert their identity in a single sign-on (SSO) operation. If specified, this is typically a long character string value known only to the domain configuration.

  Note: This is not a user account password and should not be used as such.

- **An enablement indicator**—Specifies if the domain is enabled for use at run time (the default). This allows you, for example, to configure domains for deployment while not making them available to an OpenEdge application until its deployment is complete.
- **System options**—Allows you to specify options that can be passed to authentication systems for interpretation by configured ABL authentication callbacks.
- **A tenant name**—For a multi-tenant database, identifies the tenant to which the domain belongs, and through which all users in the domain access the database. For a non-multi-tenant database, this setting does not exist.
- **Additional settings**—To support auditing and other options that currently have limited or no affect on user validation.

The following sections describe the requirements and essential features of domain configuration.

Requirements to configure a domain

Before you configure a domain:

- If the database is multi-tenant, ensure that the tenant for which the domain will be defined is already configured.

- Ensure that the authentication system required to authenticate user identities in the domain has been defined in the database.

Defining the name of a domain

A domain name must be unique for all domains defined in the database and can contain a maximum of 64 case-insensitive characters from the following restricted character set:

- a to z
- A to Z
- 0 to 9
- Any of the following seven special characters:

```
# $ % & - _ .
```

Note: The restricted character set for domain names is the same as for domain names in E-mail domains.

The domain name can also be the empty string (""), which is reserved in OpenEdge. The empty string ("") is the value of the default (blank) domain name that OpenEdge pre-defines in every OpenEdge database. In addition, OpenEdge pre-configures and reserves additional domains whose names you cannot use to configure your own domains. For more information on these pre-configured and reserved domains, including the default domain, see Pre-configured and reserved OpenEdge domains on page 35.

Defining and specifying the system type

The system type is the name of the authentication system used to authenticate all user identities defined as members of the domain. Each authentication system supports a particular set of user accounts and allows one or both of the following authentication operations on a user identity:

- **User authentication by OpenEdge** — If the authentication system supports user authentication by OpenEdge, OpenEdge automatically supports it for SSO.
- **Single sign-on (SSO) by OpenEdge** — If the authentication system **only** supports SSO by OpenEdge, an ABL application must implement the user authentication, possibly with the help of an external user account system, such as Lightweight Directory Access Protocol (LDAP), OpenID, or GoogleID.

For more information on OpenEdge authentication operations for ABL, SQL, and database command-line utilities, see How is OpenEdge Identity Managed? on page 21.

OpenEdge supports the authentication systems shown in the following table, which describes the supported user accounts and the support for OpenEdge user authentication and SSO.

Note: In some OpenEdge database administration tools, the **Domain Type** in an authentication system configuration window is the same as the **System Type** in a domain configuration window.

Table 2: OpenEdge authentication systems

Name("Domain Type")	Description	OpenEdge SSO supported?	OpenEdge User authn supported?
`_oeusertable`	Built-in system that authenticates against the `_User` table accounts, supported for all OpenEdge database clients	Yes	Yes
`_oslocal`	Built-in system that authenticates against the operating system user accounts, supported for all OpenEdge database clients	Yes	Yes
`_extsso`	Built-in system that relies on an ABL application to do user authentication, supported for ABL clients only	Yes	No
`user-defined`	User-defined system with a name you specify, supported for ABL clients only. If you enable it only for OpenEdge SSO, it relies on an ABL application to do user authentication. If you enable it for OpenEdge user authentication, you must also configure an ABL authentication callback to validate user credentials against user accounts.	Yes	Yes/No

OpenEdge support for user authentication and SSO

OpenEdge provides built-in user authentication support using the `_oeusertable` and `_oslocal` authentication systems anywhere OpenEdge accepts user credentials for authentication. This means that wherever an OpenEdge-supported user authentication operation is initiated:

1. OpenEdge identifies the authentication system from the asserted domain.
2. Delegates the user authentication operation to the appropriate security system.
3. Certifies or invalidates the security token to authorize access to the asserted domain, based on the result returned, and sealing the security token in the process.

This built-in user authentication occurs when connecting to a database or when setting the user identity for an ABL session or an existing database connection within an ABL application.

OpenEdge provides built-in SSO support within ABL applications, where a sealed security token can be used to set the user's identity for different ABL sessions and database connections. In general, if a domain is configured for built-in user authentication, it is also configured for built-in SSO. OpenEdge also provides limited user authentication support for database command-line utilities in order to access databases in certain offline states (see Pre-configured and reserved OpenEdge domains on page 35).

For the built-in `_extsso` and any user-defined authentication system that you enable for OpenEdge SSO only, an ABL application must implement its own user authentication operation to validate user credentials and seal a security token. The application can then use the sealed security token in an OpenEdge SSO operation to validate and establish the user identity for an ABL session or database connection.

For a user-defined authentication system that you enable for OpenEdge-performed user authentication, you configure an ABL authentication callback that validates the user credentials provided by an unsealed security token, which is input as part of an OpenEdge user authentication operation. Similar to an application-implemented user authentication, the authentication callback can perform the validation using any source of trusted user accounts, possibly including an external user account system such as LDAP, OpenID, or GoogleID. The callback then returns the result to OpenEdge, which certifies or invalidates the security token accordingly to authorize access to the asserted domain, sealing the security token in the process.

You can therefore effectively implement your own authentication system in ABL, which OpenEdge can use to authentic user credentials, much as it uses the built-in `_oeusertable` and `_oslocal` authentication systems. The main difference is that a user-defined authentication system only works within an ABL session that is initialized and running, and so cannot accept user credentials to connect a database on the command-line. However, it does work with user credentials specified in a `CONNECT` statement, among other OpenEdge user authentication operations in ABL.

ABL provides language elements to manage built-in OpenEdge user authentication and SSO operations, as well as to implement user authentication within the ABL application or within a user-defined authentication system enabled for user authentication using an ABL callback. For more information, see Authentication in ABL applications on page 38.

Specifying the system type

To specify a system type in a domain configuration, the authentication system must already be configured in the OpenEdge database. The built-in authentication systems are pre-configured in every OpenEdge RDBMS and cannot be deleted or changed other than to configure ABL callbacks to extend the built-in user authentication and SSO mechanisms.

You can define one or more user-defined authentication systems by specifying names and other configuration options for them in the database administration tools. For example, in the Data Administration utility, use the **Admin** > **Security** > **Domain Maintenance** > **Authentication Systems** menu option. To create a new authentication system, click **Create**, then enter its name in the **Domain Type** field and other options as required. You cannot specify a leading underscore (_) in the authentication system name, as this is reserved for OpenEdge built-in authentication systems. You can also edit an existing authentication system by selecting it in a list and changing any options that are enabled for input.

Two of the most important options to set in a user-defined authentication system configuration include the field to specify an ABL callback and the toggle box for indicating whether the authentication system supports only OpenEdge SSO operations (where a callback is optional) or supports OpenEdge user authentication operations as well (where an authentication callback is required).

You can use the ABL API for Multi-tenant Maintenance to configure user-defined authentication systems. For more information on this API and how to create and configure ABL callbacks for authentication systems, see *OpenEdge Development: Programming Interfaces.*

Entering a domain access code

The domain access code supports two related functions:

1. **Sealing a security token** — It is used to cryptographically seal the user's credentials in a security token after they are successfully authenticated for a given user identity in this domain. *Sealing* a security token identifies the user, puts in read-only mode, and establishes the trusted origin.
2. **Validating a security token** — In an SSO operation, OpenEdge matches the access code used to seal the security token to the access code defined for the domain configuration. If they match, the SSO is successful, and the user identity represented by the security token is authorized to access OpenEdge resources in that domain.

Therefore, it is important to choose an access code for the domain that is unique to both the domain and the OpenEdge installation so the security token cannot inadvertently be used for SSO operations in another installation.

Enabling and disabling domains

Enabling and disabling domains is a way for a database administrator (DBA) or a developer deploying an application to control access to data. The DBA might want to temporarily disable access for certain domains during certain maintenance operations. A developer deploying an application might want to disable all application domains until the deployment process is otherwise complete.

To accommodate these and other needs, domain configurations include the option to enable or disable domains. A newly created domain is enabled by default.

Entering system options

One use of this field might be to specify mapping information between this OpenEdge domain and an operating system domain when the authentication system is `_oslocal` on a Windows system, which supports its own domains. In this way, a Windows user might extend the built-in validation of their Windows user name with some other ABL check involving their Windows domain as well.

A user-defined authentication system with a configured ABL authentication callback might accept a variety of options, depending on the user account system that the callback relies on to authenticate user credentials.

For more information on the syntax of the options you can specify in this field, see the topic about using ABL callbacks in authentication systems in *OpenEdge Programming Interfaces.*

Identifying the tenant to which a domain belongs

In a non-multi-tenant database, there is no tenant to identify, as a non-multi-tenant database has only a single tenancy that all users share. In a multi-tenant database, you must identify a tenant for every domain that you configure. Any tenant you identify in a domain configuration must already be configured in the database. Initially, a multi-tenant database has one default regular tenant pre-configured with the name, `"Default"`. There are no super tenants configured in a database unless you create them.

In essence, tenancy provides authorization for a user to access data that is owned by a regular tenant or effectively owned by a super tenant. A user with regular-tenant authorization accesses only the tenant's own data in multi-tenant tables, as well as all the data in tables shared by all tenants. A user with super-tenant authorization can potentially access all the data in all tables, similar to a real tenant. However a super-tenant user's data access capabilities also depends on their ABL user permissions or SQL user privileges settings for table and field access, exactly any regular-tenant user.

In addition, through table and field permissions set on a domain, you can create a regular-tenant or super-tenant authorization pass with specific data access permissions for the configured tenant. You can also create as many domains for a single tenant as you need to provide the variety of data access controls that you require for the tenancy. For more information on multi-tenancy, see *OpenEdge Getting Started: Multi-tenancy Overview.* For more information on configuring tenants, see documentation on multi-tenant database configuration using the Database Administration Console. For more information on setting access permissions on domains, see Configuring and Implementing Authentication in OpenEdge on page 29.

In any case, by providing the name of a tenant in a domain configuration, all users that are members of this domain assume the configured tenancy when their user identity is authenticated for a connection to this multi-tenant database.

Note: Because the domain configuration for each multi-tenant database specifies the regular or super tenant through which a user accesses the database, when a given user identity is authenticated for a different database, the name of its regular or super tenant can also be different.

Pre-configured and reserved OpenEdge domains

OpenEdge pre-configures several built-in domains in every OpenEdge RDBMS that you cannot remove. These built-in domains include the following:

- **Default (blank) domain**—This domain is initially configured with the `_oeusertable` authentication system. You can configure it with any available authentication system, including a user-defined one. However, you cannot remove the default domain from the OpenEdge database, and you cannot change its domain name or tenant name (`"Default"`). It is provided for backward compatibility with earlier releases where users are not defined as members of a domain. So, if you migrate an application to OpenEdge from an earlier release that does not define domains, all users of the application are now defined as members of the default domain. In this case, users can identify themselves using a non-qualified user ID (see Specifying a user ID for OpenEdge authentication on page 13.
- **`WINDOWS` and `UNIX`**—These two domains are configured with the `_oslocal` authentication system and are enabled for user authentication operations against user accounts managed by the current operating system. You cannot change the configuration of these domains or remove them. OpenEdge reserves them for access to the database command-line utilities (`_dbutil` executables). These domains are not supported for use in an ABL application.
- **`WINDOWSID` and `UNIXID`**—These two domains are configured with the `_oslocal` authentication system and are enabled for SSO operations on the operating system process user ID. You cannot change the configuration of these domains or remove them. OpenEdge reserves these domains primarily for access to the database command-line utilities (`_dbutil` executables) when a database is not in an open and recovered state and the user's already authenticated operating system user identity must be used to access the database. For an ABL application, OpenEdge also assigns one of these domains for any user who connects a database without authenticating a user identity and is thus assigned the default OS process user ID as their connection identity.

Configuring authentication-enabled domains

As might be clear from the previous sections, OpenEdge cannot authenticate a user account name in all OpenEdge domains. And of those domains in which OpenEdge can authenticate users, some are restricted for use by SQL clients and database utilities, while others are available for user authentication in all OpenEdge database clients. The common requirements that allow OpenEdge to authenticate users in an OpenEdge domain define an authentication-enabled domain, regardless of which database client supports it.

Thus, an *authentication-enabled domain* is one where the domain is configured with:

- **An authentication system that is enabled for user authentication** — The **Enabled** or **Enable authentication** toggle box (depending on the tool) is selected in the authentication system configuration.
- **An authentication system that has access to a source of valid user accounts** — For a user-defined authentication system, as implemented by the configured ABL authentication callback.
- **The option to enable the domain for run-time access** — The **Domain Enabled** or **Enabled** toggle box (depending on the tool) is selected in the domain configuration.

Given all domains with this definition:

- The built-in, authentication-enabled domains support OpenEdge-performed user authentication anywhere user credentials can be specified for all database clients.
- The user-defined, authentication-enabled domains support **only** those OpenEdge-performed user authentication operations that are available as ABL statements, functions, or methods.

Configuring authentication

To configure authentication, the following tasks must be completed for all types of OpenEdge database clients and sessions:

- In multi-tenant databases, configure all required regular tenants and super-tenants. You must add a super-tenant if you want to make super-tenant access rights available for a database. Also, the tenant must be enabled for data access in order to authenticate the tenant identity. For more information, see documentation on configuring tenants using OpenEdge Management.
- Configure all required domains with their authentication system, access code, and for multi-tenant databases, their tenant (see Defining and configuring security domains on page 30). Note that you must configure at least one domain per each tenant or super-tenant to which you want to provide access in a multi-tenant database. That one domain can have the same name as the tenant for which it is configured.
- For OpenEdge-performed user authentication, configure users in the user accounts associated with each domain configured for user authentication. Note that the OpenEdge `_User` table accounts supported by `_oeusertable` domains, and the Windows user accounts supported by `_oslocal` domains are the only built-in user accounts that OpenEdge supports for access to multiple tenants and super-tenants of a multi-tenant database, because Windows supports multiple domains that can be arbitrarily mapped to tenants. Of these two choices, Windows user accounts offer the stronger overall security. Unix user accounts support only a single domain, and can therefore only authenticate a single tenant. Unix accounts can be useful in a multi-tenant database, for example, to support users of a single super-tenant domain, such as database administrators. However, using Unix accounts to authenticate a regular tenant is not recommended, because all Unix users would be limited to a single tenancy.

  Caution: The OpenEdge `_User` table accounts do not meet recommended best practices for strong user authentication. The `_User` table accounts do not support: strong password storage, password rules, password history, resistance to off-line alterations, password cracker utilities, authentication time-out, and so on. Check these factors with your production site security policies when configuring domains to use `_User` table accounts.

- An additional option when setting up `_User` table accounts is to specify that a given account is available for SQL access only, which provides another level of authentication.
- Assign users to roles responsible for managing security. For DDL administration, you assign users to the role of SQL DBA (database administrator). For general ABL security, you assign users to the role of security administrator using the database administration tools. And for auditing, you assign users to specific audit roles.
- Finally, you can prevent users with the default (blank) user ID from connecting to or accessing a database. This requires all database access settings to specify a **non-blank** user ID.

Run-time domain configuration

OpenEdge supports multiple configurable domains. At run time, all configured and enabled domains are made accessible through a trusted domain registry. A *trusted domain registry* is a run-time cache of domain entries loaded for any OpenEdge session or database connection that establishes a user identity. For SQL clients and database command-line utilities, a separate local registry is loaded for every database connection.

For ABL database clients, OpenEdge supports three basic configurations for trusted domain registries:

- **Using local database registries** — Each database connection relies on the local domain registry automatically loaded by OpenEdge for each database connection to allow a user authentication or SSO operation to set the user identity for the connection. In an ABL client, a database connection can be configured to trust a domain registry that the ABL application can load in the ABL session.
- **Loading a session registry from a database** — The ABL client uses the `LOAD-DOMAINS( )` method on the `SECURITY-POLICY` system handle to load the domains of a single OpenEdge RDBMS into client memory, creating a session domain registry. A *session domain registry* allows the ABL client, through user authentication or SSO, to set a user identity for the ABL session that is separate from any database connection identity. In addition, by setting the database administration option, **Use Application Domain Registry**, for every database that is connected in an ABL session, all the databases rely on the single source of domain configuration provided by the session domain registry. Note that the session domain registry never automatically loads. So, if no ABL session registry is available when needed, the trusted domain registry for every database connection auto-defaults to the local database registry.

  Note: If you rely on database local domain registries, you can still use a single operation to set multiple database connections to the same identity as the session. However, you must ensure that the domain and user configurations are identical for every database, **especially** the access code that you assign to the same domain in each database.

- **Creating a session registry entirely in ABL** — The ABL client uses the `REGISTER-DOMAIN( )` method on the `SECURITY-POLICY` system handle to create and load domains into the session registry one domain at a time, which includes setting the domain name and description, domain type (authentication system), and the access code for each domain. This is the only way to build a custom session domain registry that might be a subset, or even an entirely different set of domains from those configured in connected databases. This might be useful, for example, if you rely on the domain name of an authenticated session user to determine what databases to connect. However, such a custom session domain is inherently more error prone to maintain than one loaded from a database, and it is less secure, because you have to expose the clear-text access code for each domain to the application. You also cannot use an ABL-created domain registry to authenticate identity for a multi-tenant database connection, because such a registry has no tenant information associated with the registered domains.

  Note: In an ABL application, loading or creating any session registry is one of the first tasks after starting up, so user logins can be supported.

Entering user credentials in OpenEdge

When a user runs any database client, they typically enter a user ID and password as their user credentials, and are often prompted to enter the user and domain name separately. Some of these clients might also run with the default blank user ID (blank user and domain name) if the blank user ID is enabled for database access (using database administration tools) and the application otherwise supports it. However, an SQL client has no single default identity and always requires a non-blank user ID.

The following table shows the supported formats with which a user can enter a user ID that is directly interpreted by OpenEdge. These formats are necessary because OpenEdge supports multiple domains, and many users have legacy user accounts that never supported domains. If these legacy accounts are now configured with the default domain, users can enter their non-qualified user ID as they did when it had no domain configured for it.

Table 3: Formats for entering an OpenEdge user ID

This format...	Means...
No entry (internally, `""`)	The blank user in the default (blank) domain
`@`	The blank user in the default domain
`@`*`domain-name`*	The blank user in the specified domain
`user-name`	The specified user in the default domain
`user-name``@`	The specified user in the default domain
`user-name``@`*`domain-name`*	The specified user in the specified domain

Authentication in ABL applications

In addition to configuring a run-time domain registry, the primary component for performing authentication in an ABL application is the handle-based client-principal object, which is the security token for passing security information between the ABL application and the OpenEdge runtime (AVM). A client-principal object is involved with every user authentication or SSO operation in an ABL application. Either OpenEdge creates and seals the object in response to authenticating a user ID and password when creating a new database connection, or you create and initialize the object in order to authenticate and set a user identity for an existing ABL session or database connection. You can then use a client-principal previously authenticated and sealed by either your or another ABL session to set the user identity for an ABL session or database connection using SSO.

Note: A standard best practice is to have one and only **one** instance of a client-principal object for each login session.

Initializing a client-principal object for user authentication

The client-principal object that you create has a number of attributes that you can set and methods that you can invoke to initialize it for user authentication. The most important of these for an OpenEdge-performed user authentication are the `USER-ID`, `DOMAIN-NAME`, `PRIMARY-PASSPHRASE`, and `SESSION-ID` attributes. Note that the `USER-ID` attribute actually specifies the non-qualified user ID (or user name). You can initialize both the `USER-ID` and `DOMAIN-NAME` attributes using a single fully qualified user ID that you assign to the `QUALIIED-USER-ID` attribute. For more information on setting a fully qualified user ID see Entering user credentials in OpenEdge on page 38. For information on setting the `SESSION-ID` attribute, see Exporting and importing a client-principal object on page 39. Other possible initialization settings of a client-principal object include attributes to specify:

- **`DOMAIN-DESCRIPTION` attribute** - Domain description
- **`AUDIT-EVENT-CONTEXT` attribute** - Audit context information
- **`LOGIN-EXPIRATION-TIMESTAMP` attribute** - Date and time of login expiration
- **`CLIENT-WORKSTATION`, `CLIENT-TTY`, and `LOGIN-HOST` attributes** - Session login environment
- **`ROLES` attribute** - List of user role names defined in the format of a permissions list passed to the `CAN-DO` function.
- **`SET-PROPERTY( )`, `GET-PROPERTY( )`, and `LIST-PROPERTY-NAMES( )` methods** - For setting application-defined name-value pairs that can hold any useful login information

There are additional attributes for returning the status of a client-principal object and its login session, such as the `DOMAIN-TYPE`, `LOGIN-STATE`, `STATE-DETAIL`, and `SEAL-TIMESTAMP` attributes. Yet another set of attributes track a user's tenancy for multi-tenant database connections during a login session (see Managing identity for multi-tenancy on page 42).

Once you complete a user authentication operation and the object is sealed (starting a login session), you can no longer directly modify its attributes or other settings. When OpenEdge creates a client-principal as it connects to a database, it sets only the most important attributes before sealing the object. You can re-initialize a sealed object to authenticate a new login session using the `INITIALIZE( )` method, which creates a new unsealed one, but the old security token is discarded and cannot be re-used to set the user's identity. Otherwise, a login session otherwise remains active until you invoke the `LOGOUT( )` method or the `AUTHENTICATION-FAILED( )` method on the client-principal object to change its LOGIN state.

Exporting and importing a client-principal object

An important feature of the client-principal object is that it can be exported from one ABL session and imported to another using the `EXPORT-PRINCIPAL( )` and `IMPORT-PRINCIPAL( )` methods. This can be done before or after the user identity it represents has been authenticated and the object sealed.

Exporting an unsealed client-principal object, for example, allows an ABL client session to pass a user identity to an PAS for OpenEdge instance acting as an authentication service. The instance then imports the security token as its own client-principal object and after authenticating and sealing the object with a user authentication operation, it exports the sealed security token back to the ABL client to set the user identity for the session and database connections, as appropriate, using an SSO operation.

Exporting and importing a sealed client-principal allows an authenticated user identity to be passed from one ABL session to another in an multi-tier application, allowing multiple ABL sessions of one application to set the same user identity through SSO or to maintain a single application identity that is separate from the individual identities set for each session. In either case, the original user authenticating session typically initializes the SESSION-ID attribute with a unique value that identifies the sealed client-principal as representing the identity for the entire application. Depending on the session type of a PAS for OpenEdge instance, this also allows all ABL sessions in a single (virtual) application "session" to share data that is stored and keyed with this SESSION-ID value in a context database.

A useful source for the SESSION-ID attribute value is the ClientContextId property on the Progress.Lang.OERequestInfo class. The information encapsulated by this class is passed with any communication between a PAS for OpenEdge client and the instance. The object is accessible to the instance with a request from the client through the CURRENT-REQUEST-INFO attribute of the SESSION system handle. The object is also accessible to the client with a response from the instance in the RESPONSE-INFO attribute on the server object handle used to access the instance. Similarly, a client can set the ClientContextId property value by accessing the OERequestInfo object using the REQUEST-INFO attribute on the server object handle, and the instance can set a response using the CURRENT-RESPONSE-INFO attribute on its SESSION handle. By using the same unique value for the ClientContextId property an application-wide context can be maintained across many clients and instances, and the same value can be stored in the SESSION-ID attribute of an exported or imported client-principal object to keep track of the user identity for all these connected client and PAS for OpenEdge sessions. For examples and information about using the ClientContextId property to maintain a user login session across the several ABL sessions of a multi-tier application, see *OpenEdge Development: Programming Interfaces*.

Note: The real type of the object referenced by attributes such as REQUEST-INFO and CURRENT-RESPONSE-INFO is Progress.Lang.OERequestInfo. However, the object is stored in the handle as a Progress.Lang.Object. Therefore, you must cast the object reference that is the attribute value down to a Progress.Lang.OERequestInfo reference in order to access the object's class members.

OpenEdge-performed authentication and SSO

Using client-principal objects, ABL allows you to set the same or a separate user identity for each of an application's ABL sessions and database connections, as well as allowing you to maintain a separate identity for an entire ABL application. In order to set session and database connection identity, ABL supports the following mechanisms for either user authentication or SSO operations, or both:

- **AVM startup** — Optionally authenticates a user identity for each database connection by specifying the User ID (-U) and Password (-P) startup parameters on the command line or in a parameter file. The domain specified for each user ID must be authentication-enabled. For each successfully authenticated database connection, OpenEdge creates a sealed client-principal object to represent the connection identity that you can return to an ABL session using the GET-DB-CLIENT function.

  Note: This mechanism does not support domains configured with user-defined authentication systems that are authentication-enabled using ABL callbacks.

- **CONNECT statement** — Authenticates user identities for new database connections exactly as for AVM startup, but from within the ABL application.

- **SETUSERID function** - Attempts to set the user identity for a single existing database connection in a user authentication operation from a specified user ID and password. The user ID can only specify domains configured for the _oeusertable authentication system. If the user authentication is successful, OpenEdge creates an authenticated and sealed client-principal object to represent the connection identity that you can return to an ABL session using the GET-DB-CLIENT function.

Note: If successful, setting a database connection identity with this function locks out any setting of this connection identity using the SET-CLIENT( ) method on the SECURITY-POLICY system handle. SET-CLIENT( ) can be unlocked to set this connection identity by invoking SET-DB-CLIENT with the Unknown value (?) passed in place of the client-principal handle.

Note: With limited domain support, Progress Software recommends that existing applications replace calls to this function with calls to the SET-DB-CLIENT function.

- **SET-DB-CLIENT function** - Attempts to set the user identity represented by a client-principal object for an existing database connection. It performs a user authentication operation if the client-principal is unsealed and the specified domain supports it, and seals the object if the authentication is successful. It performs an SSO operation if the client-principal object is sealed.

  Note: If successful, setting a database connection identity with this method locks out any setting of this connection identity using the SET-CLIENT( ) method on the SECURITY-POLICY system handle. SET-CLIENT( ) can be unlocked to set this connection identity by invoking SET-DB-CLIENT with the Unknown value (?) passed in place of the client-principal handle.

- **SET-CLIENT( ) method** - This method on the SECURITY-POLICY system handle initially attempts to set the user identity represented by a client-principal object for the current ABL session. It performs a user authentication operation if the client-principal is unsealed and the specified domain supports it, and seals the object if the authentication is successful. It performs an SSO operation if the client-principal object is sealed. If it successfully sets the session identity, it then attempts to set the identity for every connected database by implicitly calling the SET-DB-CLIENT function in an SSO operation on each database connection.

Application-performed user authentication

OpenEdge allows you to write your own user account system in an ABL application, then use its sealed client-principal to set the authenticated user identity for ABL sessions and database connections. For domains configured with the built-in _extsso authentication system or a user-defined authentication system enabled for SSO only (see Defining and specifying the system type on page 31), you must implement the authentication system in an ABL application.

Note: Domains configured with the built-in _extsso or user-defined authentication systems, including those enabled for user authentication with ABL callbacks, cannot authenticate users for access to SQL clients or database utilities.

For application-implemented authentication:

- You handle the management of user accounts, which you must define and store securely, including all required user identification information.
- You determine the criteria for authenticating users, but the result must include an authenticated and OpenEdge-compatible user ID, and an additional unique value, that you can use to initialize the QUALIIED-USER-ID and SESSION-ID attributes of a client-principal object. If the user authentication fails, call the AUTHENTICATION-FAILED( ) method on the unsealed client-principal object to change its INITIAL state to the FAILED state, which properly registers a failed login audit event.
- After you have authenticated a user identity and initialized the client-principal object to represent it, you must manually seal the client-principal object using the SEAL( ) method, which starts a login session for

the user. This method requires the access code that is configured for the user's domain as an input parameter to cryptographically seal the security token that the object encapsulates. So, when you configure user-defined or `_extsso` domains be sure to make note of the access code that you assign to each one.

- After you have sealed the client-principal object, you can then use it in SSO operations with the `SET-DB-CLIENT` function or the `SECURITY-POLICY:SET-CLIENT( )` method to set the user identity for the ABL session and its database connections.

  Caution: When you use the `SEAL( )` method, you must protect the domain access code, which is exposed in your ABL code. To avoid this problem, use domains that are configured with a user-defined authentication system that is enabled for user authentication using an ABL callback.

- When the user login session ends, the sealed client-principal is no longer needed. Terminate the login session by calling the `LOGOUT( )` method on the object handle to change its LOGIN state to the LOGOUT state, which properly registers a logout audit event. You can then safely delete or re-initialize the client-principal object using the `INITIALIZE( )` method.

You cannot take a user identity that must be authenticated with application-implemented user authentication and authenticate it for a connection to a database in the `CONNECT` statement or on the AVM startup command line. Only OpenEdge can authenticate users to startup a database connection. So, the initial database connection must be secured with a different identity, which you can reset with an application-authenticated identity using SSO.

Caution: Application-implemented user authentication is very flexible in that you can determine what constitutes a user and the criteria for their authentication. However, it is inherently less secure than using a built-in authentication system, such as `_oeusertable` or `_oslocal`, or a user-defined authentication system that is enabled for user authentication using an ABL callback. Using authentication systems that support OpenEdge-performed user authentication, you can complete the entire user authentication in a single operation with `SET-DB-CLIENT`, `SET-CLIENT( )`, or the `CONNECT` statement, which automatically seals the client-principal using the domain access code encrypted and stored in the trusted domain registry. Your application never needs to, and cannot, access the value of this code.

Therefore, implement application user authentication **only** if you have implemented a secured code-base environment, preferably on a locked-down authentication server, especially when exposing domain access codes within your application.

Managing identity for multi-tenancy

As noted previously, when a user identity is authenticated for a multi-tenant database connection, it assumes the regular or super tenancy that is configured for the user's domain in that particular database. For a different multi-tenant database, the user's tenant or super tenant can have a different name, as specified by the domain configuration in that database.

Because an ABL application and its databases can be configured to use a single session domain registry for all database connections, the domain registry does not contain tenancy information for every database that a user might access. Instead, ABL reads the database domain configuration for each new database connection that is authenticated for a given user identity in order to identify the user's tenancy. However, ABL also caches this database tenancy information in the user's client-principal object. This allows the user to assert the same tenancy for additional connections to the database in other ABL sessions or for another connection to the same database that has been disconnected in the current ABL session, all without having to lookup the tenant identity in the database each additional time. The user thus gets consistent tenant access across the login session because the tenant ID in the client-principal is used for all SSO operations that set the user identity.

This cached tenancy information includes three components, the:

- Database logical name or alias
- Name of the tenant or super tenant
- Tenant ID for the tenant or super tenant (set as the value of the **External ID** in the database tenant configuration)

These three components help to ensure that the tenancy information for a given user identity is unique for all likely databases, even if two different databases have the same logical name. ABL maintains this cache of tenancy information in a given client-principal object for every database connection for which the object has been used to authenticate the connection identity. The client-principal object maintains this cache whether or not the database is still connected or is connected in the same ABL session where the connection was originally authenticated. The information in this cache is only removed when the client-principal object is deleted from a session.

Generally, OpenEdge identity management enables tenancy to be transparent so your code works the same whether it is running against a multi-tenant or non-multi-tenant database. However, if you have a need (perhaps to retrieve other tenancy information as a super-tenant), you can access the current database tenancy information that is cached in a client-principal object using the following attribute and methods:

- **`DB-LIST` attribute** - Returns a comma-separate list containing the logical names or aliases of all databases for which the object has been used to set a connection identity
- **`TENANT-NAME( )` method** - Returns the name of the tenant or super-tenant whose identity is configured for the user's domain in the specified database
- **`TENANT-ID( )` method** - Returns the configured tenant ID for the tenant or super-tenant identity in the specified database

Overview of Safe User ID

The OpenEdge Database includes robust multi-tenant support that manages physical access to tenant data through the means of a user identity and domain that can be controlled by an ABL application. This multi-tenant support allows an ABL application to serve multiple tenants within a single database. However, as an ABL developer, you need to make sure that access to multi-tenant data is managed properly to guarantee that tenant data remains available only to authorized users.

Each database connection has the concept of a current user identity. The database connection's current user identity on a multi-tenant database is associated with a tenancy and establishes all authorization to a tenant's private data. The database connection physically prevents access to another tenant's data until the connection’s current user identity is changed to another tenant.

In an ABL application, a client-principal object is used by the application to assert the current user identity for authorization to database tenant data. When that ABL business application is operating in a multi-session PAS for OpenEdge server, any one ABL session can be executing requests from different tenant users. The ABL application is responsible for maintaining synchronization between the database connection’s current user identity and the authenticated client identity for whom the application is executing a request.

An OpenEdge SQL database connection operates on the same principal, whereby the database connection’s current user identity is established when the SQL application client connects to the database with a user identity, an OpenEdge security domain specification, and a password. The SQL server’s database connection identity is retained for the lifetime of the connection, and only the tenant data that is authorized through the user identity can be accessed. The SQL application developer does not require synchronizing the database connection's current user identity and the tenant user for whom the database is connected.

The Safe User ID feature assists the PAS for OpenEdge's ABL application developer in maintaining the synchronization between the database's current user identity and the client for whom the tenant data is being accessed.

How coding errors can lead to tenant data bleed

The key factor in maintaining the integrity of access to tenant data is carefully managing the identity of the current user of the connection to the multi-tenant database. In a multi-tenant environment, the identity of the current user is associated with a tenancy. Managing the current user identity of the database connection regulates the physical access to tenant data that the ABL application code is authorized to access.

The phenomenon in which one's tenant's data becomes available to a user of a different tenant is referred to as **tenant data bleed**. An OpenEdge database connection prevents data bleed in most scenarios. However, if an ABL application has coding flaws in its identity management of the current OpenEdge database user identity, the ABL application can be the source where bleeding tenant data originates. Most of the ABL application data bleed scenarios stem from not correctly setting the multi-tenant database connection's user identity before accessing tenant data.

When the ABL application establishes a new current user identity in a database connection, three things occur:

1. The multi-tenant database's buffers are cleared of previously loaded tenant-specific data. Shared non-tenant tables or buffers for non-multi-tenant tables are not cleared.
2. The access to physical tenant data in the database is authorized according to the user's tenancy.
3. Access to tables and fields in the database is governed by the user's tenant authentication privileges.

The following coding errors have the potential to expose tenant data to unauthorized users:

- A safe user identity is not established for database connections at application startup.

  For internal processes, an ABL application often needs to have access to shared non-tenant data that is stored in the OpenEdge database, without inadvertently having access to any specific user's tenant data. Setting a safe user identity prevents any user identity failure from leaving the database connection in an undetermined state.

  The best practice of setting a safe user identity at database connection time, or at any time in the application's lifecycle, prevents accidental inclusion of private tenant data during the application startup or re-initialization.

- The database connection's current user identity is not initialized prior to accessing tenant data.

  When the PAS for OpenEdge ABL application session executes a client request, the user's tenant identity is retained from what was set in the previous client request. When the current request's user tenant identity and the current user identity of the database connection do not match, and the ABL application accesses the multi-tenant database, data bleed can occur.

  For example, `User1@XYZ` was the last user of the database connection in the previous client request. A subsequent request on the same connection is made by `User2@ABC`. If the user identity of the connection has not been changed to the current request user, `User2@ABC` could possibly see `User1@XYZ`'s tenant data.

  This situation can happen in an ABL application's Activate event procedure, or in any common initialization that executes at the beginning of each client request. In the event that the authentication fails, the previously connected user can be used if the proper error handling code is not provided by the ABL application.

  The best practice is for the ABL application to ensure that all code paths leading from the initiation of a client's request has initialized the database connection's current user identity, without errors, before any and all tenant data operations.

What is a safe user?

A safe user is an identity that has no tenant data access and has required access privileges to shared data. Ideally the safe user should be a placeholder identity that is defined in the user account store that is used by the PAS for OpenEdge instance in which the ABL application executes.

When an administrator wants to apply a safe user identity to a client's session because the client is running a multi-tenant application, the administrator needs to control the identity that is used when a client request arrives. The identity that is used to connect the database at client startup should be the **safe user**, because this identity is used to set the agent session for this client request and will cause the agent to be reset to this safe user identity on any subsequent requests from the client session.

The database administrator is responsible for defining the identity of the safe user and ensuring its continuous availability for the lifetime of the running ABL application.. Designating the safe user has the following requirements:

- The safe user identity will be used for all new database connection's initial user identity by using the `-U/-P` connection parameters.
- The safe user must not have access to any non-blank tenant data.
- The default blank user-id may not be used as a safe user.
- A tenant or super-tenant user may not be used as a safe user because the safe user must not have access to any specific tenant data. (Note that the Safe User ID feature is for PAS for OpenEdge servers, and running as a super-tenant is highly discouraged.)
- The safe user identity cannot be changed programmatically by the ABL application. The safe user identity is established when the database connection is made and remains unchanged for the lifetime of that connection. You can disconnect and reconnect to the database using a different identity by using the `-U` parameter, but once connected, the safe user identity remains until that database is disconnected.

When Safe User ID is configured on a multi-tenant database, the identity that is used to create the initial database connection is designated as the safe user. Therefore, the database administrator is responsible for ensuring that the safe user identity is always used when the initial database connection is created.

If you change the Safe User ID configuration in any way (for example, disable it, change the policy, and so on), that change does not take effect until you reconnect to the database.

After the Safe User ID is established by the DBA at initial database connection time, the ABL application is responsible for ensuring that it consistently uses the same safe user identity for any subsequent re-connections to the database.

For details about configuring the Safe User ID feature, see Configure Safe User ID in a multi-tenant database in *Introduction to Database Multi-tenancy*.

How Safe User ID protects tenant data

When the Safe User ID feature is enabled in a multi-tenant database, the identity of the initial database connection's current user user is automatically stored as the safe user identity and cannot be changed until the database is disconnected and reconnected. The database connection automatically sets its current user identity to this store safe user identity at either the beginning of the ABL session's next client request or end of its current client request, according to the Safe User ID feature's configuration.

By switching to a safe user at these events, any unhandled failure in establishing a new tenant user identity by the ABL application as part of the new client request cannot accidentally continue and access the tenant data using the identity retained from the previous request's tenant.

As an added safety measure, if the safe user cannot be set as the database connection's current user identity at any time, the database connection becomes locked. A locked condition does not permit access to any tenant data until a viable user identity can be set in the database connection. A locked condition can be detected by the ABL application so that it can choose the appropriate recovery process. To establish a new, viable safe user, the database must be disconnected and reconnected so that the safe user identity is the result of a database user login.

Note: The Safe User ID feature applies to only to PAS for OpenEdge clients of a multi-tenant database. Batch, GUI, character-based UI, and SQL clients do not implement this feature.

What happens during a client invocation

For each client request that is handled by an ABL application, the AVM executes any configured Activate and Deactivate event procedures at the beginning and completion of the request, respectively. The Activate and Deactivate event procedures encapsulate logic that executes automatically.

The Activate and Deactivate event procedures are executed whenever a PAS for OpenEdge agent receives a request of any kind during an agent session, irrespective of the session, application, or request type that is being executed by the agent. However, the safe user identity that is applied during the activation or deactivation of a request that is handled by an agent originates from a database connection that is initially started from a client that uses a connection to a database that is configured with the Safe User ID feature.

Coding the Activate and Deactivate event procedures in the ABL application is optional, but these procedures provide the appropriate place to perform whatever initialization, error handling, and cleanup operations are necessary with respect to executing the client request. Those event procedures also become appropriate points in a client request's execution to have the database connection set the stored safe user identity.

For details about coding the Activate and Deactivate event procedures, see Programmatically use Safe User ID on page 46.

For other best practices in protecting tenant data, see Additional coding best practices on page 48.

Programmatically use Safe User ID

In multi-tenant applications, one of the operations that always happens at the beginning of a client request is to establish the identity of a tenant user in the database connection. As a best practice, you do this by setting a valid user identity in the ABL application's Activate event procedure. OpenEdge does not support the ability to programmatically change the safe user identity for a database. The use of the Safe User ID feature does not require any modification to the ABL code. However, as a best practice, you should insert code into the ABL application's Activate event procedure to do the following:

- Check whether the database connection is locked
- Determine whether the current user identity of a database connection user is the safe user

These capabilities are useful when, in the midst of the business application execution, you need to verify whether the database connection is available. A locked database connection indicates that that safe user has failed to be authenticated, which normally requires remediation within the ABL application. The best place to check for database connection authentication failures is in the Activate event procedure.

The following topics describe the behavior that the safe user ID feature, if configured in the multi-tenant database, has on the Activate and Deactivate procedures in the ABL application.

How Safe User ID is used in the request life cycle

It might be useful to consider that a client request life cycle has three phases: activation, execution, and deactivation. With regards to asserting the safe user identity, you might design the ABL application to participate in that cycle in either the Activate or Deactivate procedures. It depends on what makes the most sense for your ABL application.

Note that coding the Activation and Deactivation event procedures in an ABL application is optional. The Safe User ID feature will ensure that tenant data is protected regardless. However, it is a best practice to implement these procedures in a multi-tenant application. The best practice is always to implement an Activate procedure, and before a client request is executed, to ensure in the Activate procedure that the database connection is not locked.

Safe User ID lockout policy

The main cause of tenant data bleed is when the ABL application fails to handle an authentication error while setting a database connection's user identity. By default, the AVM does not clear the previous user identity from the database connection. In the case of an authentication error, this default behavior permits the ABL business logic to continue and assume that the database's security is operating with the new user identity. Setting the safe user identity changes this default behavior by re-asserting the safe user identity either before or after a client request is accepted. If the authentication of a new client user fails, and a safe user identity is configured on the database, the available connection becomes automatically set to the safe user and tenant data access is prevented.

When a failure occurs in setting the database connection's user identity to the safe user identity, the *safe* behavior is to prevent the ABL business application from having access to anything in the database until a viable safe user or tenant user identity has been validated and set in the connection. To provide this safe behavior, the Safe User ID feature locks out the database connection from any access to the database by any user identity, in the event of an authentication failure of the safe user identity.

Note: Because the database connection originates with a connection using the safe user identity, an authentication failure is unlikely to occur unless the safe user identify has expired or has been revoked after the database connection was established.

To detect whether a database connection is in lockout mode, you can invoke the `GET-SAFE-USER()` method. If the method returns the `UNKNOWN ( ? )` value, then the database connection's user identity is in an indeterminate state and the database connection is locked.

To unlock a database connection and resume operations requires that one of the following operations complete successfully:

- Use the `SET-DB-CLIENT()` method and successfully authenticate a user identity in the database connection.
- Use the `SETUSERID()` method and successfully authenticate a user identity in the database connection.
- Use the `SET-CLIENT()` method on the SECURITY-POLICY system handle and successfully authenticate a user identity in the database connection (note that `SET-CLIENT()` lockout policies apply).
- Use the `GET-SAFE-USER()` method to get a copy of the sealed client-principal object that represents the safe user identity.
- Use the `SET-SAFE-USER()` method as part of exception handling, whereby if you have a critical failure in the ABL application, you may set the safe user identity to prevent the application from having access to tenant data.
- Disconnect the database and reconnect it to renew the safe user identity from a fully authenticated database user.

How to check for a locked database connection

When the Safe User ID feature is enabled on a multi-tenant database, you should design the ABL application to check for a locked database connection before processing a client request. A best practice is to do this check within the Activate event procedure by implementing the `GET-SAFE-USER()` method. If this method returns a value showing that the connection is locked, the likely cause is an authentication failure of the safe user. In the event of such a failure, you can insert code that resets the database connection user to a valid user, because you do not want to permit access to the database otherwise. For more information, see `GET-SAFE-USER()` method in *ABL Reference*.

Additional coding best practices

Another important factor in maintaining the integrity of tenant data access is the application's control of tenant data that is cached in an ABL session's local data storage. Here is where you must programmatically ensure that any cached tenant-related data is cleared in conjunction with its changing of the database connection's user identity.

Initialize tenant data stored in global variables or shared session spaces

After correctly setting the user identity, the ABL language engine clears the database connection's buffers. However, if the application code makes a copy of tenant data in global variables or any shared session space (for example, an external file or a caching service), this data could potentially be seen by another tenant that uses the same application.

When copying tenant data outside of the database, the ABL application should ensure that global or shared data is properly initialized prior to, and after, a client request is executed. The following example shows one of many ways where tenant data can be cached and subsequently cleaned up when the user identity of the database connection changes (note code in **bold**):

```
define static temp-table ttTenantEmployeeCache like mtdb.EmployeeProfile.

// make a local copy of a tenant table record
buffer-copy mtdb.EmployeeProfile to ttTenantEmployeeCache.

    …    // run the request

//   switch user-id & tenant in database connection
set-db-client(hNewClientID, mtdb).

//   clear copies of tenant record data
empty-temp-table ttTenantEmployeeCache.
```

Ensure client user identity is synchronized across the application

In an ABL application architecture where a single client's requests may execute in different AVM sessions, such as PAS for OpenEdge services, it is possible that a service's multi-tenant database connection's user identity may become out of sync with that of the client user identity. This can happen when one PAS for OpenEdge service calls a secondary service and fails to pass the correct user identity. When that condition exists, the secondary service could retrieve and use incorrect tenant data.

To prevent this situation, you should make sure that the fully qualified user identity (for example, `User1@XYZ`) that you pass to the secondary service matches the fully qualified user identity obtained from the database connection.

The following code snippet shows an example of passing the client identity to the secondary service:

```
define variable hSecondaryAppServ as handle no-undo.
define variable hCPCopy as handle no-undo.

// create server hSecondaryAppServ.
// hSecondaryAppServ:connect( … ).

// set client's user-id in the privary service's multi-tenant database
set-db-client( hClientCP, mtdb).

    …  // run the request

// call the secondary application server service with the client's user-id
hCPCopy = get-db-client( mtdb ).
hSecondaryAppServ:request-info:setClientPrincipal( hCPCopy ).
run something on hSecondaryAppServ ().
```

```
// clean up deep copy of Client-Principal obtained from the database connection.
if valid-handle(hCPCopy) then delete object hCPCopy.
```

The following code snippet shows an example of using the passed client identity in a secondary service:

```
define variable hClientCP as handle no-undo.

hClientCP = session:current-request-info:getClientPrincipal().

If valid-handle( hClientCP ) then do:
    // set client’s user-id in the privary service’s multi-tenant database
    set-db-client( hClientCP, mtdb).

    // clean up deep copy of Client-Principal obtained from the database connection.
    delete object hClientCP.
end.
```

4

Configuring and Implementing Authorization in OpenEdge

Where all OpenEdge RDBMS clients rely on a single authentication model for all of their security components, OpenEdge supports two entirely different authorization models, one for SQL, and the other for ABL and everything else. The only common element between them is that they both rely on the same forms of OpenEdge user identity to configure authorization for users. The following topics describe the basic elements of how to work with these authorization models in OpenEdge.

For details, see the following topics:

- OpenEdge authorization models
- Non-multi-tenant vs. multi-tenant authorization
- Tenant data access
- When a user's domain is available for access control
- User ID patterns as ACLs and ABL permissions checking

OpenEdge authorization models

OpenEdge supports two basic authorization models, both of which rely on different types of access control lists (ACLs):

- **SQL model** — Where all access to database resources is denied by default and must be granted to a user identity as a privilege using a GRANT statement. This privilege can later be revoked using a REVOKE statement.

For more information on the SQL model, see *OpenEdge Data Management: SQL Development* and *OpenEdge Data Management: SQL Reference.*

- **ABL model** — Where all access to database resources is granted by default, and must be specifically limited in order to deny access to particular tables and fields. These access controls can be configured by editing table and field permissions in data security using the database administration tools. These permissions can control access to a database through:

  - Database command-line utilities
  - Database administration tools
  - ABL clients

  You can also manually control access to ABL application features using the `CAN-DO` function by checking a given user ID against permissions that are similar to those used to control access to tables and fields.

Non-multi-tenant vs. multi-tenant authorization

Unlike with OpenEdge authentication, where a user can authenticate to a database connection using a fully qualified user ID, regardless of the database tenancy, a fully-qualified user ID is only recognized for authorization settings on a multi-tenant database. For a non-multi-tenant database, all user IDs and user ID patterns must be specified using only a non-qualified user ID (the user name only). This is true for both the SQL and ABL authorization models. Subsequent topics that follow describe the differences in how user IDs work in the two different cases.

The on exception to this is for database command-line utilities, which always use fully-qualified user IDs to authorize any roles or capabilities, such as audit roles or the system administrator for Transparent Data Encryption (TDE). (For more information on TDE, see *Learn about Security and Auditing*.)

Tenant data access

Both models allow you to control data access for tenant and super-tenant identities based on a user's domain. For SQL, this is determined by a combination of domain membership, domain tenancy, and the role or roles that are granted. For example, a regular-tenant user in one domain can grant privileges only to another user in the same domain. However, an SQL user who is granted the DBA role can grant privileges to users in any domain, and therefore with any tenant identity.The actual granting and revoking of SQL roles and privileges is done for tenant data access using lists of fully qualified user ID's.

In ABL, a single ACL can deny or grant a table or field access privilege to all users of a domain or to specific users of a domain, and therefore to specified users with a given tenant identity, and this access (as for all data access) must be configured by a user who is a security administrator. For some examples, see Table 9: Fully qualified user ID matches on page 56.

When a user's domain is available for access control

The name of a use's domain is available for authorization purposes depending on the type of database client, the type of access, and whether the database is multi-tenant or non-multi-tenant (non-tenant). In general, authorization to access a multi-tenant database is always controlled using a fully qualified user ID, including the domain name. However, authorization to access a non-tenant database is always controlled using only the user name—non-qualified user ID without the domain name—even if the user has been authenticated in a specified domain using a fully qualified user ID.

The following table shows more specifically for what database client and what type of access you can use a fully qualified user ID or only the user name to control access to database resources, where *user@domain* represents a fully qualified user ID and *user-name* represents non-qualified user ID.

Table 4: Where a domain name is available for access control (authorization)

This database client type...	For database utility access uses...	For tenant data access uses...	For view access uses...	For table access uses...	For field/column access uses...
Database command-line utilities	*user@domain*	*user@domain*	–	*user@domain*	–
ABL non-tenant	–	–	–	*user-name*	*user-name*
ABL multi-tenant	–	*user@domain*	–	*user@domain*	*user@domain*
SQL non-tenant	–	–	*user-name*	*user-name*	*user-name*
SQL multi-tenant	–	*user@domain*	*user@domain*	*user@domain*	*user@domain*

Access to restricted database command-line utilities is granted to two user roles:

- ABL Security Administrator
- SQL DBA

In general, an ABL Security Administrator is granted access to:

- Restricted database command-line utilities
- Restricted ABL database configuration API's and classes
- Management of auditing roles
- Management of table and field time permissions checking at compile time or run time
- Restricted meta-schema tables, such as `_User`, `_File`, `_Db`, and so on

An ABL Database Administrator must take all these access permissions into account when determining what users they assign to the Security Administrator role.

An SQL DBA is granted access to:

- Restricted database command-line utilities
- Management of security for views, tables and fields

All of the database client types perform run-time authorization to tenant data, and always use a fully-qualified user ID to do this type of access control.

User ID patterns as ACLs and ABL permissions checking

ABL permissions for table and field access test a given user ID against a permission string, which is a comma delimited list of permission entries. Permission testing is performed in the ABL client at different times, all using the same permission checking algorithm:

1. Compile-time permission checking is performed when an ABL source (`.p`/`.i`/`.w`/`.cls`) file is compiled into r-code.

   Note: Compile-time permission checking no longer is a recommended best practice data security model when a database can execute pre-compiled modules from any source.

2. R-code execution checks permissions to access:
   a. Meta-schema and schema tables at run time
   b. User data tables when run-time permission checking is enabled
   c. User data tables accessed through dynamic buffers and queries

The matching rules for permission checking are simple, but powerful:

- The first match of the user ID against one of the pattern entries ends the check and returns the results to the caller.
- When no match between the user ID and any pattern entry exists, permission is always **denied**.
- If the pattern entry begins with an exclamation mark ("!"), permission is **denied** when a match between the user ID and pattern entry exists.
- If the matched pattern does not begin with an exclamation mark, permission is **granted** when a match between the user ID and the pattern entry exists.

Patterns affecting ABL permissions

The following table shows how a user ID match to denying or granting patterns affects ABL permissions.

Table 5: Granting or denying ABL permissions

If the user ID matches...	This pattern...	Grant/Deny permission...
No	`!`*`pattern-entry1,pattern-entry2`*	Deny
Yes	*`pattern-entry`*	Grant
Yes	`!`*`pattern-entry`*	Deny

The patterns contained in tables and field permission entries support wildcard characters '*' and '.', for multi-character matches and single character matches respectively. The following table lists examples of supported wildcard patterns.

Table 6: Wild cards in user ID patterns

This pattern...	Matches...
	The blank user ID
*	All user ID's
mark	The user ID, "mark"
mark*	All user ID's that begin with "mark"
*jones	All user ID's that end with "jones"
db*user	All user ID's that begin with "db" and end with "user"

The ABL permissions can be configured either in a **deny** authorization model or in an industry standard **grant** authorization model. A **deny** model is when all users are **granted** permission and specific users are **denied** permission. This **deny** authorization model is no longer recommended best practice as its default is to grant a new user account access unless otherwise denied. The **grant** authorization model is the industry recommended type, and in this model a user account is **denied** access until explicitly granted. The **grant** model is stronger as it denies new user accounts access until the DBA explicitly grants them access. The basic forms of ABL permission lists for the two authorization models are as follows.

To **deny** permissions, use this pattern:

```
!user-id-pattern[,!user-id-pattern]...,*
```

To **grant** permissions, use this pattern:

```
user-id-pattern[,user-id-pattern]...
```

As illustrated above, non-multi-tenant databases use non-qualified user ID's for permission checking, while multi-tenant databases use fully qualified user ID's. OpenEdge extends its ABL permission checking to use either non-qualified or fully qualified user ID's at all times. In this way, databases with existing user ID patterns continue to work while a database is being converted to multi-tenancy and back. During conversion, all data is moved to the default tenant, so existing non-qualified user IDs continue to work.

The user ID's and ID patterns used for permission checking change as follows:

- The user ID is split into two fields: the user name and the domain name (minus the '@' domain delimiter). If the user ID does not contain a domain name delimiter, the blank domain name is assumed. The following table shows some examples.

Table 7: User ID changes for DB multi-tenancy conversion

Source user ID	User name used for pattern matching	Domain name used for pattern matching
`"rjones"`	`"rjones"`	`""` (blank domain name)
`"rjones@acme.com"`	`"rjones"`	`"acme.com"`

- The ID pattern entry is split into two patterns: the user name pattern and the domain name pattern (minus the `'@'` domain delimiter). If the pattern does not contain a domain name delimiter, the blank domain name is assumed. The following table shows some examples.

Table 8: ID pattern changes for DB multi-tenancy conversion

Source pattern entry	User name used for pattern matching	Domain name used for pattern matching
`"rjones"`	`"rjones"`	`""` (blank domain name)
`"rjones@acme.com"`	`"rjones"`	`"acme.com"`
`"*"`	`"*"`	`"*"`

- For a pattern match to occur between a user ID and a pattern entry:
  - The pattern entry contains the single wildcard pattern `'*'`, which always matches both the user name and the domain name.
  - A match must exist between the user ID's user name and the pattern entry's user-name pattern **and** between the user ID's domain name and the pattern entry's domain-name pattern.

Caution: The `CAN-DO` function treats `'@'` as a domain delimiter by default, but this behavior can be changed by setting the `CAN-DO-DOMAIN-SUPPORT` attribute on the `SECURITY-POLICY` handle to `FALSE` or by using the `-nocandodomain` startup parameter. For more information, see *OpenEdge Development: ABL Reference* and *OpenEdge Deployment: Startup Command and Parameter Reference*.

The use of wildcards in patterns produces some equivalencies between non-qualified and fully qualified user IDs and ID patterns. For non-qualified user ID patterns, full qualification is implied:

```
"*" == "*@*"
""  == "@"
"mark" == "mark@"
```

The following table shows some examples of fully qualified user ID matches.

Table 9: Fully qualified user ID matches

This pattern...	Matches...
	The blank user name in the blank domain

This pattern...	Matches...
*	All user ID's
mark@acme.com	The "mark" user name in the "acme.com" domain
mark	The "mark" user name in the blank domain
mark*	All user names that begin with "mark" in the blank domain
*jones	All user names that end with "jones" in the blank domain
db*user	All user names that begin with "db" and end with "user" in the blank domain
@acme	The blank user name in the "acme" domain
*@acme	All user names in the "acme" domain
@	The blank user name in the blank domain
mark@*	The "mark" user name in any domain
@	Any user name in any domain
mark*@acme.*	Any user name that begins with "mark" in any domain that begins with "acme."
@.admins	Any user name in any domain that ends with ".admins"

Glossary

Glossary

Access control
See *user authorization*.

Access control list (ACL)

A list of user IDs or user ID patterns, associated with a unit of data and one or more operations on that data, that grant or deny access to the specified data and operations. Similar lists are used to assign users to roles, especially data security and administration roles.

For example, in SQL, an ACL consists of a list of user IDs associated with a specific data access privilege that is granted or denied. In ABL, an ACL consists of a list of user ID patterns that might grant specified access permissions to some users, including groups of users, and deny the same access permissions to other users or groups of users. In both cases, the established user identity is tested against the appropriate ACL when it attempts to access a controlled resource.

Access controls

Electronic records that contain the information used in the *user authorization* process. Usually, access controls record what users, groups, and roles can perform what operations on a given set of resources (such as files, database records and fields, network connections, and so on).

Access token

See *security token*.

Authentication

The process performed by an *authentication system* to validate a user's asserted identity and determine their abilities and rights to access data sources.

Authentication system

A subsystem, within a *security system*, designed to validate the identity of a user prior to granting access to resources. Validation can be in the form of *user authentication* or *single sign-on (SSO)*. Inputs to authentication mechanisms include passwords and intelligent tokens. Upon successfully completing authentication, the authentication system issues a *security token*, which can be used by other security systems in the same *security domain*. Inputs to SSO mechanisms include security tokens generated by authentication systems in the same security domain.

Authn

See *user authentication*.

Authorization

The process of determining an authenticated user's rights or permission to use a system resource. Also known as the process of granting access.

Authorization system

A subsystem, within a security system, designed to perform *authorization*. Inputs include the user's *security token* (containing their *identity* and rights) and a configuration that holds the list of protected resources and what user identities or rights may access them.

Authz

See *user authorization*.

Cipher

A software algorithm used for *encryption*, for example, DES, AES, RSA, Blowfish, and so on.

Clear text

The original data value before it is *encrypted*, or the original data value after it is *decrypted*.

Data integrity

The process of detecting when binary data has been inappropriately altered or corrupted.

Data-integrity seal

A *message digest*, *digital signature*, or *HMAC* that is attached to a block of data to provide validation of the block's *data integrity* and originator. The seal dates back to when the state of data-integrity technology was based on wax seals that were placed on paper documents. The wax seals were embedded with an imprint that was unique to the person who generated the document. Examples in computer technology include a database record, disk file, etc.

Data privacy

Data access restricted to named users while at rest or in transit. Combination of technologies:

- *User authentication* (for *authorization*)
- *User authorization* (data-*access controls*, including *access control* to the data's *encryption key*
- *Encryption* (of the data)

Decrypt

To use *encryption* to transform *encrypted data* into *clear-text data*.

Digest

A cryptographic hash algorithm. See also *message digest*.

Digital signature

An electronic signature that also includes *data integrity*. It is comprised of a message digest, to supply the data integrity, which is encrypted using an asymmetric *cipher* and the signer's private *encryption key* to provide the authenticity.

Disallow Blank UserId Connections

An option offered in the **Database Options** of the OpenEdge Data Administration utility's dialog box. When selected, this option allows an ABL OpenEdge database to be connected only after authenticating a non-blank user ID (including the domain) using the User ID (`-U`) and Password (`-P`) client connection parameters.

Domain

Shortened form of *OpenEdge security domain*.

Domain registry

See *registry*.

Encrypt

To use encryption to transform clear-text data into encrypted data.

Encrypted data

Data that has been encrypted using a *cipher* specification, optional IV, and encryption key

Encrypted private data

encrypted data who's cipher algorithm, cipher-mode, IV, and encryption key information is restricted to entities who have been explicitly granted access

Encryption

The algorithmic *obfuscation* of binary data.

Encryption algorithm

See *cipher*.

Encryption key

A binary value used to make the *encrypted data* output from an *encryption* operation appear random.

IV

Initialization vector, an optional binary value that is used in conjunction with a second binary [key] value to produce a two-factor *encryption key*. Not all *cipher* specifications allow the use of an IV. (Theory: the IV and key are stored, owned, and access-controlled as separate entities so that an attacker must break into two locations to obtain the full encryption key.)

HMAC

A form of *data integrity seal* that combines a *message digest*, for *data integrity* checking, with a secret key value that provides validation of the message's originator. An HMAC is used to exchange messages between two points, where the receiver must be capable of validating who the author of the message is and that the message has not been tampered with (i.e., altered). This form of security works when each author has a unique secret key value (often a password or passphrase) and shares their key with each message's receiver.

Identity

See *user identity*.

Login

See *user login*.

Login credentials

See *user credentials*.

Login session

See *user login session*.

Logout

Voluntary surrendering, and invalidation, of a user's login *security token*, therefore surrendering their ability to have a system or application perform any more operations for them.

MD5

Message Digest #5, which is a *digest* algorithm developed by RSA Corporation that produces a fixed length 16-byte *message digest*.

Message digest

A fixed length binary output of a *digest* (cryptographic hash) algorithm, which in theory produces a unique bit pattern for every unique byte sequence given to it. Commonly used in *data-integrity* checking. See also *MD5* and *SHA-1*.

Obfuscation

A software algorithm that scrambles data in order to hide it from others. The algorithm scrambles the data in such a way that it can be unscrambled and the original value recovered. Many times, obfuscation refers to a very light-weight (fast), low security, algorithm used for short-term storage, such as in-process memory. Full-blown *encryption* algorithms are much more heavy-weight (slow), complex, high-security algorithms used for long-term storage, such as in a database.

OpenEdge domain

See *OpenEdge security domain*.

OpenEdge security domain

A collection of one or more ABL session, ABL OpenEdge database connection, SQL 92, and database command-line utility *security systems* that share the same configuration and *OpenEdge security token*. Each member's security system loads and caches its configuration into a registry for run-time *user authentication* and *SSO* operations.

OpenEdge security token

An OpenEdge, proprietary implementation of a *security token*. It can be transported between [distributed] ABL session and/or ABL OpenEdge database connection *security systems* for the purpose of *SSO*.

PAM

See *Plug-in Authentication Module*.

Plug-in Authentication Module

An internal or external software module that provides an abstraction layer between an *authentication system* and individual *user account system* APIs. Each user account system has a purposed plug-in module. Which plug-in module is used by the *authentication subsystem* at run time comes from the authentication system's configuration. Each time support is added for a new user account system, a new plug-in module is written to support it.

Registry

An *authentication system's* run-time cache of OpenEdge *domains* and *Plug-in Authentication Module (PAM)* configurations. Each instance of the *security systems* for ABL sessions, ABL OpenEdge database connections, SQL 92 servers, and database command-line utilities has its own private registry cache. The input cached into a registry originates from the `_Sec-Authentication-*` tables contained in an OpenEdge database.

Seal

See *Data-integrity seal*.

Security domain

A security domain is considered to be an “application, or collection of applications, whose *security systems* are configured to issue and/or trust a common *security token* for the purposes of performing *authentication*, *authorization*, and/or session management.”

Security system

A security system consists of a combination of hardware and/or software that limits the exposure of application data, a computer, or a computer network to attack from intruders. Security systems generally include subsystems that consume external configurations and perform *user authentication*, *authorization*, and auditing operations.

Security token

A binary collection of information about a user's *identity* and security that is created and issued by an *authentication system* and is used by an *authorization* system to authorize access to data sources. Examples of a security token's contents include user roles, user IDs, group IDs, and other information that can be used to control access to content.

A secure and *data-integrity* protected collection of *user identity* information, including the user's account information, capabilities, rights, privileges, roles, groups, and the identity of the authentication system or application that can authenticate (or has already authenticated) the user's identity. An security token is a transportable block of data that can be used as proof of user identity by any systems or applications that have a *trust relationship* with the originator of the security token.

SHA-1

Secure Hash Algorithm # 1, which is a *digest* algorithm developed for the United States government that produces a fixed length 20-byte *message digest*.

Signature

A unique pattern, identifiable with a single person or thing, that is used for proof of authenticity. See also *digital signature*.

Single sign-on (SSO)

The process of requiring an end user to login to any member of a *security domain* one time, then being able to access any other member of the security domain without having to login again. The SSO process takes the *security token* resulting from a successful *authentication* by an *authentication system*, transports it to any other *security system* within its security domain, validates it as coming from a trusted member of the security domain, then uses it to establish the authenticated *user identity* for *authorization*.

Trust relationship

A logical connection that is established between *security domains* so that the rights and privileges of a user (asserted through a *security token*) can be validated by, and shared with, the other security domains. (A trust relationship allows users to login once and gain access to resources in other security domains without having to be authenticated again.)

Trust Application Domain Registry

An option offered by the OpenEdge Data Administration utility's **Database Options** dialog box. When set, this option causes an ABL OpenEdge database connection's *authentication system* to use the ABL session's *domainregistry* instead of its own private registry. This enables multiple OpenEdge databases to share the same registry *domain* and *Plug-in Authentication Module* configuration.

User account

An electronic record that represents a single computer system or application user's *identity*. This identity includes a unique account identifier (such as a user name) and one or more, shared secrets (such as passwords, fingerprints, digital signatures, and so on) that can be used during a *user authentication* operation to validate that the user account can only be used as proof of identity by the user it represents.

User account system

Encapsulates *user account* storage, administration tools, and a library of programmable APIs used by *authentication systems* or *Plug-in Authentication Modules* for *authentication*.

User authentication

Often abbreviated as *authn*, the process performed by a computer system or application to validate that a user's *login credentials* match the information recorded in a named *user account*. The result of successful user authentication is a sealed *security token*.

User authorization

Often abbreviated as *authz*, the process of validating a user *security token* and using its *access controls* to regulate what operations the user can perform.

User credentials

A collection of information that includes a *user account* name, shared secrets (passwords, etc.), and other (possibly identifying) information, which is submitted by a user during a *user authentication* process. Also referred to as *login credentials*.

User identity

Otherwise known as a user ID, the validated *user account* name that is unique to an individual. User identity names can be non-qualified or fully qualified. Non qualified names include only the user account name as stored in its *user account system*. Fully-qualified user identities include the user account name and the name of the [security] *domain* in which to perform *authentication*. Normally, domains are delimited from the user account name by a single character: OpenEdge uses the `'@'` as a domain delimiter.

User login

A process of identifying one's self, by submitting *login credentials* to a computer system or application as proof of *identity* to a *user authentication* process. The system or application must successfully complete the user authentication before it will run applications or operations on one's behalf.

User login session

A period of time after a security token is sealed during which it represents a valid user identity. A user login session is typically auditable for key events, such as the initial login, the final logout, and other audit events that trace changes of state in the security token between the two.