Progress®



OpenEdge® Getting Started:
Core Business Services - Security and Auditing

OpenEdge®

Copyright

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

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

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

February 2019

Last updated with new content: Release 12.0

Updated: 2019/03/01

Table of Contents

Preface....13
Purpose....13
Audience....14
Organization....14
Additional resources for further information....15
Using ABL documentation....16
References to ABL compiler and run-time features....16
References to ABL data types....16
Typographical conventions....16
Examples of syntax descriptions....18
Long SQL syntax descriptions split across lines....19
Complex syntax descriptions with both required and optional elements....20
OpenEdge messages....20

Part I: Overview....21

Chapter 1: OpenEdge Core Business Services....23
Security services....23
Identity management....24
Connection and data security....25
Auditing service....26
Run-time auditing....27
Audit trails....27
Common auditing approach....28
Audit policies....28
Audit security....28
Querying and reporting....29
Archiving audit data....29
Enabling and disabling auditing....29
Transparent Data Encryption....30

Part II: Security....33

Chapter 2: Security in OpenEdge....35
Application development....36
ABL applications....36

Open Client applications....36
Application deployment....37
Secure source code....37
WebClient applications....37
Report Builder user authentication....37
Application portability....37
Data management....38
OpenEdge levels of data security....38
Database security....38
DataServer security....39
Application feature authorization....39
ABL run-time procedure authorization....39
OpenEdge reporting: Query/Results....39
Progress Dynamics....40
Application service security features....40
AppServer program security....40
Security for OpenEdge Web services....41
Consuming external Web services in ABL....41
Data privacy for client connections to an AppServer....41
Firewall compatibility....41
Integration service security features....42
Progress Dynamics security features....42
Setting basic security options....43
Defining Progress Dynamics application security options....43
Using the Progress Dynamics security managers....44
Understanding the Repository security structures....44
Application network security using SSL....44
Auditing security....45
Core user authentication and authorization....45
Configuring OpenEdge security domains in an OpenEdge RDBMS....45
Creating user IDs in the OpenEdge RDBMS _User table....46
Authenticating and authorizing a user identity....46
Asserting a user identity that is validated using single sign-on (SSO)....47
SSL Security....47
Changing the cryptographic protocol, ciphers, and certificates 48
Adding Server Name Indication (SNI) 56

Chapter 3: Cryptography....59

Basic cryptography....59
Algorithms....60
Keys....60
Symmetric (secret) key cryptography....60
Encryption/decryption algorithms....60
Encryption/decryption modes....61

Encryption/decryption key size....61
Combine algorithm, mode, and key size....61
OpenEdge usage....62
Password-based encryption (PBE)....63
PBE algorithms....63
OpenEdge usage....63
Asymmetric (public) key cryptography....64
Asymmetric keys....64
Public-key algorithms....64
OpenEdge usage....65
Message digests....65
Message digest algorithms....65
OpenEdge usage....66
Managing cryptographic data....66
Objectives....66
Requirements....66
OpenEdge solutions....67

Chapter 4: Public-Key Infrastructure (PKI)....69

Core security services in a PKI....69
Authentication....70
Confidentiality....70
Integrity....70
Cryptography in a PKI....71
Symmetric-key cryptography....71
Message digests and MACs....71
Public-key cryptography....71
Password-based encryption (PBE)....72
Trust relationships and supporting mechanisms....72
PKI trust model....72
Digital certificates and certificate store....73
Digital signatures....75
Client and server interactions....76
Key and certificate management....76

Chapter 5: Secure Sockets Layer (SSL)....77

SSL standards support in OpenEdge....77
Features and services....78
SSL session components....78
Session identity....79
Keys and certificates....79
Algorithms (ciphers)....79
Session caching....80

Support for trust....80
SSL interactions and the user....80

Chapter 6: SSL in OpenEdge....83
OpenEdge SSL architecture....83
OpenEdge SSL client and server components....85
Non-OpenEdge SSL client and server components....85
Using SSL in OpenEdge....86
Managing SSL server identity....86
Establishing SSL server identity....87
Using the default SSL server identity....87
Managing your own SSL server identity....88
Configuring and running SSL sessions....88
Configuring SSL server sessions....89
Configuring SSL client sessions....91
Accessing an SSL servers X.500 Subject Name....94

Part III: Auditing....95

Chapter 7: Auditing in OpenEdge....97
OpenEdge auditing....97
What you can audit....98
Evaluating your auditing requirements....99
Application developer decisions....100
End user responsibilities....100
OpenEdge auditing recommendations....101
Auditing OpenEdge databases or clients earlier than Release 10.1....101
Audit data storage management....101

Chapter 8: Audit Security....103
Overview....104
Asserting user identity....104
Adding security through separation of duty....104
Managing audit privileges....105
Inheriting audit privileges: the security administrator....105
Assigning audit security privileges....105
Creating a primary and a secondary audit administrator....106
Granting audit privileges....107
Revoking audit privileges....108
Audit data archival security....108
OpenEdge Audit Archiving....108
Audit security for database clients, tools, and utilities....109

Audit security for OpenEdge databases....109
Audit policy security....109
Sealing the audit data....110
Resolving audit policy conflicts....110
Exporting audit policy configuration as an XML file....112
Choosing additional audit options....112
Configuring additional user authentication systems and domains....112
Controlling the user ID and login sessions in audit data records....113
Audit archiving....115
Archiving audit data....115
Assigning a database a unique identifier....116

Chapter 9: Configuring OpenEdge Auditing....117
Getting started....117
Enabling an OpenEdge database for auditing....118
Connecting to the database and assigning audit-related privileges....119
Using Audit Policy Maintenance for policy creation....119
Setting up OpenEdge auditing context....119
Auditing context architecture....120
Recording auditing context information 121
How auditing context is referenced by audit event records....122
Adding auditing context to the application....123
Managing auditing context....123
Migrating an existing application to use auditing....124

Chapter 10: Developing an Audit-enabled OpenEdge Application.127
Developing an audit-enabled application....127
Audit-enabling an OpenEdge database....127
Defining application audit events....128
Audit-enabling your OpenEdge application....128
Implementing additional auditing options....128
Updating an existing ABL application with auditing....131
Audit-enabling your SQL application....131

Chapter 11: Deploying an Audit-enabled OpenEdge Application...133
Overview....133
Developing the company's audit policy....134
Configuring auditing at the production site....134
Setting up audit administration....134
Loading audit event definitions....135
Loading predefined audit policies....135
Setting up audit data access....135
Applying audit policy for production systems....135

Writing custom audit reports....135
Preparing to deploy your audit-enabled application 136
Upgrading an existing application to use auditing....136

Chapter 12: Maintaining Audit Data....139

Common audit data maintenance tasks....139
Backing up and restoring an audit-enabled database....139
Archiving and loading audit tables....139
Copying an audit-enabled database....140
Recovering audit data....140
Modifying the indexes generated for OpenEdge audit tables....140
Modifying the audit table storage area....140
Disabling auditing....140
Handling long-term storage growth....140
Returning audit data from offline storage....141
Ensuring audit data integrity....141
Auditing performance....141
Run-time audit maintenance....141
Monitoring your databases health....141
Querying audit data....142

Chapter 13: Querying and Reporting on Audit Data....143

Overview....143
Audit data schema overview....144
Audit data querying and reporting....145
Reporting committed data only....145
Internationalization considerations....145
Reporting on a central archive database....145
Reporting on a single database with multiple GUIDs....145
Reporting from multiple audit databases....146
Deactivated audit indexes....146
Reporting event descriptions....146
Optional client session information....146
Audit data event context 147
Audit data application context....148
Logically grouping related audit data in an audit event group....149
Grouping audit data by transaction ID....149
Grouping audit data by database connection 150
Reporting additional audit event details....150
Reporting by audit date and time....150
Reporting by user ID....150
Reporting modified field old/new values....150
Identifying database events....152

Internal audit event policies....153
Efficient index use....153
Query guidelines....155
Reporting on audit data with prefiltered and custom reports....156
Generating a custom filtered report....159

Part IV: Transparent Data Encryption....163

Chapter 14: Transparent Data Encryption in OpenEdge RDBMS...165

Defining the problem space....165
Providing data privacy....166
Defining Transparent Data Encryption....166
Data security....167
Data Storage....167
Encrypted Data Configuration....168
Encryption Key....168
Before you start....169
What is encryptable....170
Data considerations....171
Transparent Data Encryption feature summary....171
Enabling Encryption....172
Adding an encryption area....172
Executing the ENABLEENCRYPTION command....173

Chapter 15: OpenEdge Key Store....177

General key store basics....177
OpenEdge key store....178
Configuring key store access....179
Manual start....179
Autostart....180
Key store maintenance....181
Backups....181
Modifying passphrases....181
Rebind....182
Key store reconstruct....183

Chapter 16: Configuring Transparent Data Encryption policies....185

Creating encryption policies....185
Creating encryption policies with PROUTIL EPOLICY....186
Creating encryption policies with the Data Administration tool....186
Encrypting your existing data....189
EPOLICY MANAGE UPDATE example....189

Encryption of new data....190
Maintaining transparent data encryption....191
Rekeying encryption policies with the Data Administration tool....192
Rekeying an encryption policy with PROUTIL EPOLICY....193
Changing the cipher of an encryption policy with PROUTIL EPOLICY....194
Viewing encryption policy history in the Data Administration tool....194
OpenEdge SQL support for transparent data encryption....195
Using the CREATE TABLE statement....195
Adding an encrypted column....196
Using the CREATE INDEX statement....196
Using the ALTER TABLE statement....197
Viewing encryption policy with the SHOW ENCRYPT statement....198
Deleting encrypted objects with OpenEdge SQL....198
Disabling encryption....198

Appendix A: Audit Data Tables....201
Audit data table schema....201

Appendix B: Preconfigured Audit Policies....235
Using the preconfigured OpenEdge audit policies....235

Index....245

Preface

For details, see the following topics:

- Purpose
- Audience
- Organization
- Additional resources for further information
- Using ABL documentation
- Typographical conventions
- Examples of syntax descriptions
- OpenEdge messages

Purpose

OpenEdge® supports two core business services: security and auditing. Security comprises several different functions, such as authentication, authorization, confidentiality, integrity, trust, identity management, and auditing. Auditing is a core business service in its own right, providing you with the means to track who did what, where, when, and how.

This manual provides an overview of security features supported by OpenEdge and all of its components, with references on where to find more information about them. OpenEdge provides security features that affect access to individual components and features. It also provides support for features of a Public-Key Infrastructure (PKI) that you can use to secure communications among OpenEdge components on a network. This manual provides a primary point of reference for information on PKI and how OpenEdge implements features of a PKI as part of its overall security support.

The manual also provides information about identity management in OpenEdge and an overview of auditing features supported by OpenEdge, such as auditing security, developing and deploying an audit-enabled application, and maintaining, querying, and reporting on audit data.

Audience

Anyone who is interested in OpenEdge security, including architects, developers, and IT administrators will find useful information in this manual to both plan and implement security for an application. Additionally, anyone who is interested in implementing an auditing solution will find details regarding planning for, implementing, and maintaining audit policies and data.

Organization

OpenEdge Core Business Services on page 23

Defines and provides an overview of the core business services, security and auditing, supported in OpenEdge.

Security in OpenEdge on page 35

Provides an overview of all security features supported by OpenEdge, with references to other OpenEdge documentation.

Cryptography on page 59

Defines cryptography and describes how it is supported in OpenEdge.

Public-Key Infrastructure (PKI) on page 69

Provides an overview of PKI, including information on how OpenEdge supports PKI features and where to find more information about them.

Secure Sockets Layer (SSL) on page 77

Provides an overview of SSL, the primary implementation of PKI features that OpenEdge supports.

SSL in OpenEdge on page 83

Describes the OpenEdge SSL architecture and how each SSL-supported OpenEdge component works within it.

Auditing in OpenEdge on page 97

Describes the OpenEdge auditing core service and provides details about planning to implement an auditing solution.

Audit Security on page 103

Provides details about various aspects of auditing security: managing privileges; archiving audit data records; audit security for database clients, tools, utilities, and audit policies; and configuring additional user authentication systems and domains.

Configuring OpenEdge Auditing on page 117

Describes how you and your customer sites can get up and running with auditing.

Developing an Audit-enabled OpenEdge Application on page 127

Provides an overview of how to develop and audit-enable an OpenEdge application.

Deploying an Audit-enabled OpenEdge Application on page 133

Provides details about audit-enabled OpenEdge application deployment.

Maintaining Audit Data on page 139

Describes common audit data record maintenance tasks.

Querying and Reporting on Audit Data on page 143

Describes how to query and report on data in the audit tables.

Transparent Data Encryption in OpenEdge RDBMS on page 165

Defines and describes Transparent Data Encryption in the OpenEdge RDBMS.

OpenEdge Key Store on page 177

Defines and describes the OpenEdge key store. The key store is an critical part of Transparent Data Encryption.

Configuring Transparent Data Encryption policies on page 185

Describes how to create and maintain encryption policies for Transparent Data Encryption.

Audit Data Tables on page 201

Describes all the audit data tables.

Preconfigured Audit Policies on page 235

Describes each of the preconfigured audit policies included with OpenEdge.

Additional resources for further information

You will also find information related to establishing security and auditing solutions for your database and applications in the following other manuals and help systems:

- *OpenEdge Data Management: Database Administration*
- *OpenEdge Data Management: SQL Development*
- *OpenEdge Data Management: SQL Reference*
- *OpenEdge Development: Basic Database Tools*
- *OpenEdge Deployment: Managing ABL Applications*
- *OpenEdge Development: Programming Interfaces*
- *OpenEdge Development: ABL Reference*
- Data Administration online Help
- Data Dictionary online Help
- Audit Policy Maintenance online Help

Using ABL documentation

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

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

References to ABL compiler and run-time features

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

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

References to ABL data types

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

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

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

Typographical conventions

This documentation uses the following typographical and syntax conventions:

Convention	Description
Bold	Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.
Italic	Italic typeface indicates the title of a document, or signifies new terms.
SMALL, BOLD CAPITAL LETTERS	Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, **GET** and **CTRL**.
KEY1+KEY2	A plus sign between key names indicates a **simultaneous** key sequence: you press and hold down the first key while pressing the second key. For example, **CTRL+X**.
KEY1 KEY2	A space between key names indicates a **sequential** key sequence: you press and release the first key, then press another key. For example, **ESCAPE H**.
Syntax:	
`Fixed width`	A fixed-width font is used in syntax, code examples, system output, and file names.
`Fixed-width italics`	Fixed-width italics indicate variables in syntax.
`Fixed-width bold`	Fixed-width bold italic indicates variables in syntax with special emphasis.
`UPPERCASE fixed width`	ABL keywords in syntax and code examples are almost always shown in upper case. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class.
Period (.) or colon (:)	All statements except `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` end with a period. `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` statements can end with either a period or a colon.
[ ]	Large brackets indicate the items within them are optional.
[ ]	Small brackets are part of ABL.
{ }	Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.
{ }	Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.
|	A vertical bar indicates a choice.
. . .	Ellipses indicate repetition: you can choose one or more of the preceding items.

Examples of syntax descriptions

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

Syntax

```
ACCUM aggregate expression
```

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

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

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

Syntax

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

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

Syntax

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

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

Syntax

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

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

Syntax

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

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

Syntax

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

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

Syntax

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

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

Syntax

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

Long SQL syntax descriptions split across lines

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

In this example, CREATE VIEW is followed by several optional items:

Syntax

```
CREATE VIEW [owner_name.] view_name
  [ ( column_name[, column_name] ... ) ]
  AS [ ( ] query_expression[ ) ] [ WITH CHECK OPTION ] ;
```

Complex syntax descriptions with both required and optional elements

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

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

Syntax

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

OpenEdge messages

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

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

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

- Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.
- Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

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

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

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

I

Overview

For details, see the following topics:

- OpenEdge Core Business Services

1

OpenEdge Core Business Services

OpenEdge® provides core business services that support all levels of application development, deployment, and management. A *core business service* is functionality built into core components of the OpenEdge platform (OpenEdge RDBMS and ABL Virtual Machine (AVM)) that support common business services required by many enterprise applications. A *common business service* can include any number of business functions shared by a variety of applications that are typically provided by the application developer. Examples of common business services include user login, application event logging, and data integrity checking.

For details, see the following topics:

- Security services
- Auditing service
- Transparent Data Encryption

Security services

Security services consist of several core services that support the following basic security functions:

- Identity management on page 24
- Connection and data security on page 25

The following sections describe these basic security functions and how they are supported in OpenEdge. For more information on particular OpenEdge components and the specific security features that they support, see Security in OpenEdge on page 35.

Identity management

Identity management provides services that identify users of a software system, validate user credentials to help verify that users who seek to access the system are trusted users, and validate a users identity against access controls that verify their rights to access protected system resources and thereby permit or prevent access to these resources. The basic identity management services include:

- Authentication on page 24
- Authorization on page 24

Authentication

Authentication is the process of verifying that a users claims to a software system represent a valid proof of the users identity, which the software system can then assign to itself for the performance of its operational functions. For example, when a user logs into an application, an authentication system ensures that the users login credentials identify a user that can operate the application before the application will run. If the login credentials identify a valid user, the application runs as an agent of the user, which allows the users identity to extend to application components that might also be distributed in secure relation to one another across a network.

The minimum requirement to authenticate a user is a user account system, a user account in that system identified by a *user ID* (or *user name*), and a shared secret (typically a *password* or *passphrase*) that only the user account system and the owner of the account know. The authentication system accesses the user account system and asks it to verify that the user's account exists, that the account is usable, and that the secret entered by the user matches the secret known to the user account. Only if the user account system successfully validates the users account does the authentication system establish the *user identity* for an application. Otherwise, the application fails to run on behalf of the specified user.

In OpenEdge, the authentication of user identities is supported through configurable user authentication systems. These authentication systems include the OpenEdge RDBMS, which can maintain its own set of valid user accounts, and support for other user account systems that OpenEdge or an OpenEdge application can use to authenticate user identities. These authentication systems also support a mechanism to assert a previously authenticated user identity as the current application or database connection identity. For more information on OpenEdge support for authentication, see Security in OpenEdge on page 35 and *OpenEdge Getting Started: Identity Management*.

Authorization

Authorization grants or denies access to protected resources based on the established user identity. Thus, an authorization system acts, based on the results of the authentication system, to authorize access to restricted application features and data. *Permissions* describe what actions a user identity or *role* can perform on a protected resource. User roles allow multiple user identities to comprise a group that shares the same permissions to protected resources. Once authenticated, a given user identity automatically inherits the permissions associated with the identity and any user role (or roles) to which it is assigned. For example, a security administrator role might allow any user who is assigned that role to manage user accounts and all other security functions in an application.

In OpenEdge, authorization using permissions is supported in ABL (Advanced Business Language) to match users against capabilities defined in the OpenEdge RDBMS, both at compile-time and run-time. OpenEdge also supports a different model of authorization based on privileges for SQL database clients. OpenEdge supports user roles in the OpenEdge RDBMS, in auditing, in the Web Services Adapter (WSA), and in OpenEdge Explorer and OpenEdge Management. For more information on OpenEdge support for authorization, see Security in OpenEdge on page 35 *OpenEdge Getting Started: Identity Management*.

Connection and data security

Connection and data security services include a wide array of mechanisms designed to secure and control access to data, both for static and dynamic data storage, and during the transport of data over a network or over a dedicated connection between system components. The purpose of these services is to achieve predictable levels of confidence in the following attributes of system data:

- Confidentiality on page 25
- Integrity on page 25
- Trust on page 26

Confidentiality

Confidentiality (also referred to as *data privacy*) assures that data communicated between any two or more entities can only be read by the entities for whom the data is intended. In order for data privacy to work effectively, each party in the communication must have a means to write and read the data that is readily available to the intended parties, but is not available at all to unintended parties; and it must work with intended parties who do not necessarily know about each other before hand. In addition, the time between the writing and the reading of the confidential data must be acceptable to both parties.

Confidentiality is primarily a concern over a network where client and server components of a distributed application are exchanging data. The mechanisms used to ensure data privacy rely largely on various forms of *cryptography*. Cryptography is a means to encode data so that only components that share an otherwise secret means to decode the encoded data can read it.

The cryptographic mechanisms used to ensure data privacy are generally chosen to balance security with performance. OpenEdge supports a variety of cryptographic mechanisms in ABL for application development and in the network infrastructure to connect the distributed components of a deployed Internet or intranet application. OpenEdge also supports Transparent Data Encryption for the OpenEdge RDBMS, which is whole-database cryptographic system to support a highly secure database environment. For more information on:

- **Transparent Data Encryption** — See Transparent Data Encryption on page 30
- **OpenEdge support for cryptography** — See Cryptography on page 59
- **OpenEdge support for data privacy in a distributed application** — See Public-Key Infrastructure (PKI) on page 69

Integrity

Integrity assures that data has not been altered (intentionally or unintentionally through bit errors) between a given origin in time or space and a given destination in some other time or space. Mechanisms, such as Cyclic Redundancy Codes (CRCs), exist to protect data from certain types of accidental or intentional bit errors during data transmission.

However, other means are necessary to protect data against more intentional or higher-level alterations to the data. These mechanisms typically include message digests and message authentication codes (MACs). A message digest, in particular, provides a value that can be compared before and after a message is sent to verify that the message has not been altered, and a MAC is a message digest with a key to make that message abbreviation secure. For auditing, MACs are used to secure the contents of audited data.

OpenEdge supports message digests and MACs for various components and services, including the AVM and OpenEdge RDBMS. These mechanisms are related to cryptography. For more information, see Cryptography on page 59.

Trust

Trust is the ultimate foundation of all security systems, because ultimately, a security system is as secure as the trustworthiness of those who have the power to access and override that security system. While security systems attempt to assure that entities are properly authenticated and authorized, and that data is confidential, reliable, and nonrepudiatable, the entire enterprise depends on trust.

The question of trust is a whole study unto itself, and it is really quite independent of the technology required to support a security system. The definition of trust provided by the ITU-T Recommendation X.509 specification is:

‘Generally, an entity can be said to "trust" a second entity when it (the first entity) makes the assumption that the second entity will behave exactly as the first entity expects.

Ultimately, trust depends on the confidence that users have in the trust mechanisms and all of the nontechnological entities involved with them that are designed to support trust in a security system.

Trust mechanisms typically implement a model (*trust model*) for facilitating trust in a security enterprise. Many trust models have been developed from which to implement the relationships (*trust relationships*) among the various entities in a security infrastructure. OpenEdge implements trust mechanisms that support the trust model provided by Public-Key Infrastructure (PKI), a methodology for supporting security services throughout an enterprise network.

Among the components of a distributed application, one or both of the client and server identities of two communicating components can authenticate one another in order to verify the validity of their peer on the other end of the connection. They can also authenticate data exchanged between them and thereby provide nonrepudiation of the data exchanged.

In data communications, *nonrepudiation* for a sender ensures that the entity that sends a message cannot later deny having sent it, and for a receiver, ensures that the entity that has received message cannot later deny having received it. Nonrepudiation for message senders is often supported by digital signatures, and for receivers by an audit of the message receipt (see Auditing service on page 26).

OpenEdge also supports the authentication of server component identities to corresponding client components in distributed network applications using the Secure Sockets Layer (SSL), which is an implementation of a Public-Key Infrastructure (PKI). For more information on:

- **PKI** — Public-Key Infrastructure (PKI) on page 69
- **SSL** — Secure Sockets Layer (SSL) on page 77
- **Support for SSL in OpenEdge** — SSL in OpenEdge on page 83

The PKI trust model relies on various authorities and agreements that certify entity interactions and that software must support in order to implement a PKI. For more information on:

- **The trust model supported by PKI** — Public-Key Infrastructure (PKI) on page 69
- **The elements of PKI implemented by OpenEdge** — See Public-Key Infrastructure (PKI) on page 69 and Secure Sockets Layer (SSL) on page 77

Auditing service

The OpenEdge auditing core business service is a fully integrated, highly configurable, secure solution that includes auditing for the OpenEdge database, database utilities, ABL and SQL applications. Because auditing can consume large amounts of resources, you audit only the events you want, and to the level of detail you specify, by creating policies.

To help you get started with auditing, OpenEdge includes several predefined OpenEdge audit policies, which you can import and use to begin collecting audit data. For more information, see Getting started on page 117 and the Audit Policy Maintenance tool online help.

The OpenEdge auditing solution includes:

- Run-time auditing on page 27
- Audit trails on page 27
- Common auditing approach on page 28
- Audit policies on page 28
- Audit security on page 28
- Querying and reporting on page 29
- Archiving audit data on page 29
- Enabling and disabling auditing on page 29

Run-time auditing

Auditing is the action of producing a trail of secure and nonrepudiatable events that transpired during the execution of a business process in order to detect how the process has been followed or changed, or the process data has been corrupted. Thus, where authentication and authorization manage which application users and components can access what features and data, auditing provides a history of application and data events that can be used to validate that all audited users and components and their actions were both anticipated and legal in the context of the application and its environment.

For example, if an otherwise valid user manages to exploit a security hole in an application to access forbidden features and data, auditing can record the user, the accessed data, and even the exact procedures, functions, and specific actions in the application that were executed to access the data.

Auditing relies on other basic security services to achieve its end, which is to capture application and data events in a way that cannot be altered after the fact. Even attempts to make such changes to auditing data can themselves be securely audited. Auditing is considered a basic component of application security because its results must be verifiable and unalterable. Thus, it is auditing that provides the means to ensure the nonrepudiation of internal data transactions and other application events, such as the receipt of messages. The assurance that audit data cannot be changed provides the means to verify the parties that have been involved in audited events.

The entire auditing capability is a complex one that, like security itself, involves a broad range of OpenEdge features and components. As such, it encompasses an entire core business service in OpenEdge.

Audit trails

Audit trails tell you who did what, when, where, and how, in the form of collected audit data. The audit data is stored in standard OpenEdge database tables, which makes it easy to query and report on the data. You can assign a globally unique database ID (GUID) to each OpenEdge database either when you create the database or later; if the audit data records are moved to an archive, you can use the GUID to identify in which database the audit data originated.

The audit policy is the configuration that OpenEdge components use to know what to place in the audit trail. OpenEdge places data in the audit trail only when the policy directs it to do so. No auditing is turned on in the policy until the administrator expressly turns it on.

Common auditing approach

OpenEdge auditing implements a common solution for database utilities and for both ABL and SQL clients. Any type of client access to the application data is audited in a consistent manner, whether through ABL, SQL, or a database utility.

Audit policies

Audit Policy Maintenance enables you to create and maintain audit policies—from simple to more complex—for application events, database events, and internal system events. You create audit policies that are used at run time to define exactly the audit data you want to record; you choose which events to audit and to what level. (You can also audit changes to the audit policy if you enable policy changes as an auditable event.) Once you create a policy and enable audit events, you check for and resolve any policy conflicts you find, activate the policy, commit the policy changes to the database, and begin recording audit data.

OpenEdge includes several preconfigured policies to help you get started with auditing. You can import the policies and use them as they are or modify them to suit your auditing needs. You fine-tune auditing through the polices you create. You control how much audit data is generated (and how much audit processing occurs); if you do not need to audit a particular event, simply do not enable it for auditing. Through Audit Policy Maintenance you can also import and export audit policy configurations in `XML` file format between systems and use for backup purposes. For details about what auditable events are included in the preconfigured policies, see Preconfigured Audit Policies on page 235.

Audit Policy Maintenance allows you to manage the following capabilities:

- **Auditing with multiple policies** — You can create and activate more than one policy in the same database. If more than one policy is in effect at run time, the policy settings are aggregated.

  You can segment your policy into multiple parts that are common to all databases, unique to a single database, or comprised of events that can be turned on or off at the same time. You can also define multiple policies in a database to enable switching between policies to accommodate different operational conditions.

- **Resolving audit policy conflicts** — Since you can create and activate more than one policy at once in the same database, it is possible that a conflict between policies might develop. For this reason, OpenEdge implements a set of rules for handling audit policy conflict resolution. Make every effort to check on and resolve conflicts before you commit the changes to the database and activate the policies.

  For details about the conflict resolution rules, see Resolving audit policy conflicts on page 110.

- **Easy performance tuning** — You control performance and space use by selecting which events to audit and to what level, along with which policies you want to activate and deactivate, using Audit Policy Maintenance. The more events you audit, the more audit data you generate and the greater the performance overhead. By being able to selectively control what you are auditing, you can better manage processing overhead and potential audit data storage concerns.

For more information about audit policy creation and management, see the Audit Policy Maintenance Help. You can access the help either from within Audit Policy Maintenance or by selecting **Start** > **Programs** > **OpenEdge** > **Help** > **Application Help**.

Audit security

An essential component to auditing success is the knowledge that the generated audit data is secure and protected from outside tampering. The OpenEdge auditing solution allows you to determine which authenticated users have access to audit policy configuration and audit data management, including the truncation, deletion, or archiving and loading of audit data, by assigning these specific predefined audit privileges: audit administrator, application audit event inserter, audit data archiver, and audit data reporter.

Granting of audit privileges occurs within Data Administration (in Windows) or character Data Dictionary (in character mode) for ABL administrators or through the SQL GRANT statement for SQL administrators.

To detect when an audit data record has been tampered with at the binary storage level by an unsecure or unregulated program, you can optionally seal the audit data records by using either a message digest or a message authentication code (MAC). Both allow detection of unauthorized changes if someone has attempted to modify audit data outside of an ABL or SQL application. The MAC is a message digest with a secret key, so it is more secure than the message digest.

You can also assign each database instance its own unique identifier, which provides a way to uniquely associate a database with its audit data, no matter where the audit data is archived.

For information about audit data security, see Audit Security on page 103.

Querying and reporting

Because the audit data is stored in standard OpenEdge database tables, you can easily query against the tables contents as long as you know the database schema. You can search by user ID, event ID, date, context, transaction, audit group, database connection, or authenticated client session.

You can also report on audit data using one of the built-in audit trail indexes, and using either a prefiltered report or a custom filtered report. The prefiltered reports are organized by category, such as audit policy, database schema, or user accounts, and allow you to specify a date range or audit event ID range, where to output the report (terminal, printer, XML file, or text file), and whether you want the contents to appear in a detailed or a summary format. You can also generate a custom report based on criteria you choose.

For a description of the audit table schema, see Audit Data Tables on page 201. For more information about querying and reporting, see Querying and Reporting on Audit Data on page 143.

Archiving audit data

Progress Software Corporation recommends that the audit data archiver use an archive utility to move audit data to long-term storage; this is especially important if the audit data tables grow past a certain size. You can use the audit archive utility included with OpenEdge, or you can develop your own archive utility. For more information about the audit data archiver privilege, see Audit Security on page 103.

If you manually remove the audit data from the OpenEdge database, you can preserve the data integrity by configuring the audit policy to include a MAC signature on each recorded audit event record. This allows you to validate the integrity of audit data at any time when you are archiving data. OpenEdge uses HMAC technology to preserve data integrity. The sealing of the audit data is done automatically during normal archival operations. You can further enhance security by using seals on individual records.

OpenEdge captures database row-change data to the database in which the table or field is defined. To preserve performance and storage, Progress Software Corporation recommends that you move the audit data to a specially designated OpenEdge database that has enhanced security, size, and reporting capability.

Refer to *OpenEdge Data Management: Database Administration* for more information about the audit archive utility included with OpenEdge. For information about developing your own audit archive utility, see *OpenEdge Development: Programming Interfaces*.

Enabling and disabling auditing

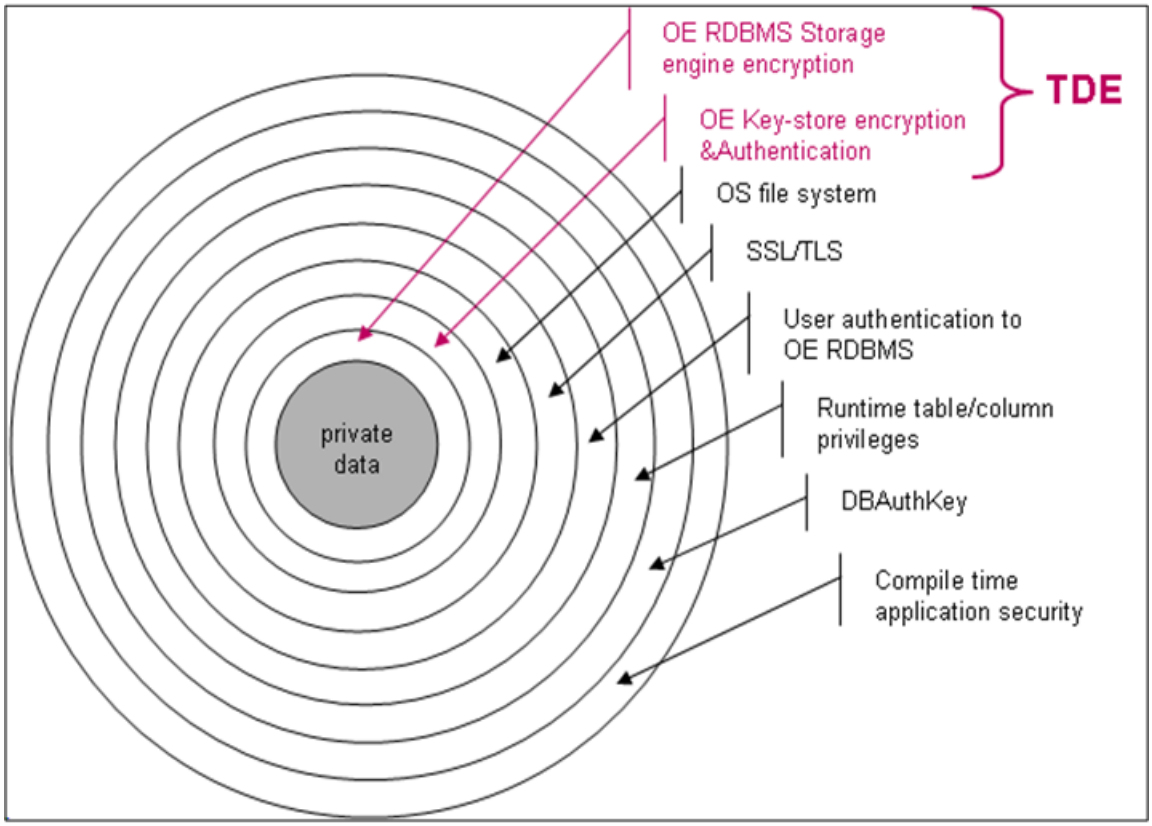
Once you audit-enable a database, the schema required for auditing is created and the database begins looking for audit policies to implement. No generation of audit data occurs, however, until you create an audit policy (to audit the table and field events) or import the preconfigured policies, commit the changes to the database, and activate the policy.

To temporarily suspend capturing audit data, you can deactivate the audit policies. Use the `disableauditing` command-line option **only** for situations in which you want to totally and **permanently** disable all auditing on a database.

Transparent Data Encryption

As part of an overall security strategy, Transparent Data Encryption provides for data privacy while the data is "at rest" in your OpenEdge database. There are many security layers in an OpenEdge application, and Transparent Data Encryption represents the inner-most layer, as shown in the following figure:

Figure 1: Security layers in an OpenEdge application



Controlling access to private data while "at rest" (that is, stored on disk inside your database), is the core of OpenEdge Transparent Data Encryption. Support for Transparent Data Encryption is embedded within the OpenEdge RDBMS and all language clients. OpenEdge combines various cipher algorithms and encryption key lengths, secure storage of encryption keys, and user access controls to your encryption keys to ensure that your datas encryption cannot be reversed by anyone other than those granted access.

Transparent Data Encryption provides protection against intruders that attempt to access your private data. Intruders fall into two categories: internal and external. An *internal intruder* is an employee or contractor who misuses granted access permissions. An internal intruder can be a system administrator who accesses your database even though they are not a database administrator or security administrator. An *external intruder* is someone who attempts to access your data from outside your company.

External intruders may try to breach your companys network security or obtain a copy of your database through other methods. Possible external intruders include a person who:

- Has accessed a companys internal network via virus or botnet on a VPN on WAN/LAN

- Obtains a stolen computer with a copy of your database
- Obtains a stolen copy of backup media
- Obtains a user name and password through deception, such as impersonating an employee and requesting a password be changed

If an intruder obtains a copy of your encrypted database, they cannot retrieve the data protected by encryption without also having the encryption keys.

Each encrypted database has a single, unique Database Master Key (DMK). The DMK is created and managed by your database administrator, and stored in your database key store, which is separate from your database. Your *key store* is an independent and secure entity that provides secure storage of data encryption keys and controls access in the form of user accounts.

Encryption of your database objects is managed through encryption policies. You define which objects are encrypted and the encryption cipher for the object. Policies are stored in your database in a designated Encryption Policy Area. No language client can query the Encryption Policy Area. Object policies utilize virtual data encryption keys derived from your DMK and the specified cipher. The encryption key for each encrypted database object is unique.

More information on Transparent Data Encryption can be found in the following locations:

- For details on Transparent Data Encryption concepts, see the following chapters:
  - Transparent Data Encryption in OpenEdge RDBMS on page 165
  - OpenEdge Key Store on page 177
  - Configuring Transparent Data Encryption policies on page 185
- For details on database administration activities related to Transparent Data Encryption, see *OpenEdge Data Management: Database Administration*.
- For details on implementing and maintaining Transparent Data Encryption with OpenEdge SQL, see *OpenEdge Data Management: SQL Development* and *OpenEdge Data Management: SQL Reference*.
- For details on implementing Transparent Data Encryption in conjunction with OpenEdge Replication, see *OpenEdge Replication: User Guide*.

II

Security

For details, see the following topics:

- Security in OpenEdge
- Cryptography
- Public-Key Infrastructure (PKI)
- Secure Sockets Layer (SSL)
- SSL in OpenEdge

2

Security in OpenEdge

OpenEdge provides features that support major security applications including government requirements, company security policies, and competitive edge. This chapter describes these features by major IT function and OpenEdge product area (with some intentional redundancy) and indicates where to go for more information about them.

For details, see the following topics:

- Application development
- Application deployment
- Data management
- Application feature authorization
- Application service security features
- Integration service security features
- Progress Dynamics security features
- Application network security using SSL
- Auditing security
- Core user authentication and authorization
- SSL Security
- Adding Server Name Indication (SNI)

Application development

OpenEdge provides security features that form the functional foundation for implementing security in both ABL and Open Client applications.

ABL applications

For ABL application development, OpenEdge supports the following security features from directly within ABL:

- User authentication through user ID and password validation to support authorized access to databases and other security features. For more information on user authentication features, see Core user authentication and authorization on page 45.
- Data privacy using the Secure Sockets Layer (SSL) between ABL client sockets and supported socket servers, between supported socket clients and ABL server sockets, between the ABL clients and databases, and between ABL clients and AppServers. For more information on SSL in OpenEdge, see Application network security using SSL on page 44.
- Compile-time authorization (the OpenEdge default) to ensure that only authorized users can compile procedures that perform specific database table and field accesses.
- Optional run-time authorization to ensure that only authorized users can run specific precompiled procedures and to otherwise provide feature authorization by defining activities-based security validation; also to ensure that only authorized users can run procedures that perform specific database table and field accesses. You can also set an authorization key in an OpenEdge database using the `DBAUTHKEY` option of the OpenEdge PROUTIL utility to prevent unwanted precompiled procedures that have already passed compile-time authorization from running against the database.
- Cryptography to ensure that unauthorized users cannot read or change data. For more information on cryptography and the cryptographic features of ABL, see Cryptography on page 59.

  Caution: Progress Software Corporation recommends that you use the cryptographic features of ABL **only** if you have a well-grounded understanding of cryptography and its usage. Use of cryptography without the necessary preparation can result in permanent data loss. In general, cryptography can have significant negative impact on application performance and decrease effective data compression for data stored in a database.

- Auditing for OpenEdge databases, database utilities, SQL applications, and ABL applications that records a non-repudiatable audit trail of both pre-defined database events and ABL-defined (or SQL-defined) application events. For more information, see Auditing security on page 45.

For more information on authorization, see Core user authentication and authorization on page 45. For more information on security features of ABL, see *OpenEdge Development: Programming Interfaces*.

Open Client applications

For .NET and Java™ Open Client applications, OpenEdge supports features that:

- Access the AppServer using SSL, both over the Internet (HTTPS) and over an intranet. For more information on SSL in OpenEdge, see Application network security using SSL on page 44.
- Secure .NET Open Client applications using different run-time packages that support a choice of strong-named assemblies, digitally-signed assemblies, or a combination of the two. For more information, see *OpenEdge Development: .NET Open Clients*.

For more information on Open Client security, see *OpenEdge Development: Open Client Introduction and Programming*.

Application deployment

Security features for application deployment include:

- Secure source code on page 37
- WebClient applications on page 37
- Report Builder user authentication on page 37
- Application portability on page 37

Secure source code

Using OpenEdge tools, you can encrypt source code that you deploy for an ABL application. Among other advantages, this allows the application to use late compilation, which allows procedures to be compiled based on preprocessor directives determined by the deployment environment without fear of unauthorized source code tampering at the user site.

For more information on using encrypted source code, see *OpenEdge Deployment: Managing ABL Applications*.

WebClient applications

WebClient™ applications are ABL applications that are deployed with a version of the AVM that can be downloaded over the Internet or an intranet in a manner commonly used by Application Service Providers (ASPs) as well as Software as a Service (SaaS) Providers. The security features available for a WebClient application include:

- Application files that can be digitally signed for deployment and download from a Web server using HTTPS
- Download of application files over secure Internet or intranet connections using SSL
- Authentication to the ASP server using user names and passwords
- Single sign-on and security caching

For more information on WebClient security options, see *OpenEdge Deployment: WebClient Applications*.

Report Builder user authentication

The Report Builder provides a means to add security to user passwords with the table interface by providing a means to specify them in an encoded form. For more information, see *OpenEdge Reporting: Report Builder Deployment*.

Application portability

For application portability, there are considerations for table and field security based on user IDs. For more information, see *OpenEdge Deployment: Managing ABL Applications*.

Data management

For data management, OpenEdge supports a security feature set involving several functional levels that largely correspond to the levels of security available for ABL application development (see Application development on page 36). OpenEdge also provides additional options specifically for database, DataServer, and SQL Server security.

OpenEdge levels of data security

The supported levels of data security include:

- Database connection authorization using SSL (see Database security on page 38).
- Compile-time authorization (the default), which requires ABL procedures and classes to have authorization to compile against a particular database. For information on compile-time authorization, see *OpenEdge Deployment: Managing ABL Applications*.
- Run-time authorization (optional), which includes setting an option to check OpenEdge RDBMS table and field permissions in ABL procedures and classes at run time, using the `DBAUTHKEY` configuration option of the PROUTIL utility to require run-time permissions checking of ABL procedures and classes that have previously passed compile-time permissions checking, and using the Type of Server to Start (`-ServerType`) startup parameter to restrict the type of server (SQL or ABL) that accesses the OpenEdge RDBMS. For information on run-time authorization for ABL procedures and classes, see *OpenEdge Deployment: Managing ABL Applications*, on the `DBAUTHKEY` option, see *OpenEdge Data Management: Database Administration*, and on the Type of Server to Start (`-ServerType`) startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.
- Cryptography, which ensures that unauthorized users cannot read or change data. For more information on cryptography and the cryptographic features of ABL, see Cryptography on page 59.

  Caution: Progress Software Corporation recommends that you use the cryptographic features of ABL **only** if you have a well-grounded understanding of cryptography and its usage. Use of cryptography without the necessary preparation can result in permanent data loss. In general, cryptography can have significant negative impact on application performance and decrease effective data compression for data stored in a database.

- Transparent Data Encryption (TDE) to provide data privacy while the data is "at rest" in an OpenEdge database. For more information, see Transparent Data Encryption in OpenEdge RDBMS on page 165.

For more information on authorization in OpenEdge, see Core user authentication and authorization on page 45.

Database security

The OpenEdge RDBMS provides the following security features:

- Identity management for database utilities, and SQL and ABL database servers.
- Setting up compile-time and run-time database, table, and field authorization. For more information on managing ABL for this purpose, see *OpenEdge Deployment: Managing ABL Applications*.
- Schema trigger security using r-code CRCs. For more information, see *OpenEdge Deployment: Managing ABL Applications*.

- Assigning an ABL database security administrator, creating internal database user accounts, and obtaining a report of users who are authenticated to use the database. For information on the administration tools available for this purpose, see *OpenEdge Development: Basic Database Tools* (using character mode) and the Data Administration online help (using GUI mode).
- Using SQL to create SQL users and assign DBAs (SQL database administrators).
- Authorizing access to SQL tables and columns by granting, verifying, and revoking privileges.
- Sealing auditing data using a MAC key. For more information on MAC keys, see Cryptography on page 59. For more information on using MAC keys to seal auditing data, see Sealing the audit data on page 110.
- Working with operating system security.
- Configuring the database server as an SSL server to enable SSL connections from ABL and SQL clients. For more information on SSL in OpenEdge, see Application network security using SSL on page 44.

For more information on security for the OpenEdge RDBMS, see *OpenEdge Data Management: Database Administration*.

For more information on authorization and authentication in OpenEdge SQL, see Core user authentication and authorization on page 45. For more information on SQL data security, see *OpenEdge Data Management: SQL Development*.

DataServer security

For DataServers, OpenEdge provides the ability to pass user credentials to foreign data sources for connection authentication. For more information, see the appropriate DataServer manual:

- *OpenEdge Data Management: DataServer for Microsoft SQL Server*
- *OpenEdge Data Management: DataServer for Oracle*

Note: OpenEdge does not return a security token associated with foreign database connections and relies completely on the foreign database authentication systems for its user authentication and privileges. OpenEdge does not support SSL client connections to DataServers.

Application feature authorization

OpenEdge provides features to authorize access to specified procedures or functional parts of an application in general ABL code, in OpenEdge reporting applications, and especially in Progress Dynamics®.

ABL run-time procedure authorization

OpenEdge provides ABL functions designed to work with permissions tables that allow you to authorize user access to specific precompiled procedures. For more information, see ABL applications on page 36 and Core user authentication and authorization on page 45.

OpenEdge reporting: Query/Results

OpenEdge Query/Results supports its own features for security management, including:

- User management and site administration for both application and data security. For more information, see *OpenEdge Reporting: Query/Results Administration and Development*.

- Database options to secure:
  - Sign-on program/product name
  - Contents of a user directory
  - Module permissions
  - Query permissions
  - Exiting an application

  For more information, see *OpenEdge Reporting: Query/Results for UNIX*.

Progress Dynamics

Progress Dynamics®, the OpenEdge tool for building late-binding application components, provides a rich set of standard feature-oriented security options that are organized for both individual user and user group authentication. For more information, see Progress Dynamics security features on page 42.

Application service security features

Application service security features primarily protect access to AppServer-oriented OpenEdge functionality, described here with respect to:

- Application service implementation on the AppServer itself
- Access to the AppServer through OpenEdge Web services
- Data privacy over the connection between the AppServer and its clients
- Internet-enabled AppServer compatibility with firewalls

For more information on AppServer application services and how they work in OpenEdge, see *OpenEdge Getting Started: Application and Integration Services*.

AppServer program security

OpenEdge supports application implementation of the following features for securing an application running on the OpenEdge AppServer:

- Authenticating users to the AppServer at connect time.
- Authenticating and transporting an application user identity across the ABL sessions of an n-tier application, and maintaining user identity context for both session-managed and session-free applications. For information on transporting (exporting and importing) user identity, see *OpenEdge Development: Programming Interfaces*.
- Authorizing user or client access to AppServer session procedures using access control lists implemented using the `EXPORT( )` method on the `SESSION` system handle.
- Using an AppServer to intelligently filter access to database resources for clients (instead of simply "passing through" client access to database tables and fields).
- Generating AppServer activity audit trails.
- Using run-time compilation and other dynamic resources to dynamically configure and run AppServer procedures according to application security constraints. This feature can also make use of encrypted source code to secure the source for run-time compilation (see Secure source code on page 37). You might also

want to implement secure AppServer application services using the completely dynamic application environment provided by Progress Dynamics (see Progress Dynamics security features on page 42).

For more information on AppServer program security, see *OpenEdge Application Server: Developing AppServer Applications*.

Security for OpenEdge Web services

For OpenEdge Web services, OpenEdge authenticates users and clients of the Web Services Adapter (WSA) through JSE security (depending on the Java Servlet Engine) in order to:

- Access WSDL files hosted by the WSA
- Run Web service applications supported by a WSA instance
- Perform WSA administration functions

For more information on security for OpenEdge Web services, see *OpenEdge Application Server: Administration*.

Consuming external Web services in ABL

OpenEdge supports secure access to external Web services in ABL using:

- SSL connections to Web servers
- User ID and password support for WSDL and Web service access

For more information on ABL support for secure consumption of external Web services, see *OpenEdge Development: Web Services*.

Data privacy for client connections to an AppServer

OpenEdge supports secure connections to the AppServer using SSL from any client. For application clients (ABL or Open Clients), this includes both Internet (HTTPS) access using the AppServer Internet Adapter (AIA) and direct intranet access using an SSL variation of the AppServer protocol. For AppServer adapters, OpenEdge also supports SSL AppServer protocol connections to the AppServer on the intranet. For more information on SSL in OpenEdge, see Application network security using SSL on page 44.

Note: AppServer agents, as ABL database clients, can also access the OpenEdge RDBMS using SSL.

Firewall compatibility

For Internet-enabled AppServers using the AIA, OpenEdge supports the following options to maximize compatibility with firewalls:

- TCP/IP port management
- Network Address Translation (NAT) and distributed configurations
- NameServer client port range value settings
- Direct connection to the AppServer without a NameServer to avoid UDP conflicts with firewalls

Integration service security features

OpenEdge supports two middleware adapters that provide messaging and integration services to OpenEdge applications. The OpenEdge Adapter for SonicMQ® (SonicMQ Adapter) allows ABL applications to communicate with other applications using the Java Message Service (JMS) infrastructure managed by SonicMQ.

The OpenEdge Adapter for Sonic ESB® (Sonic ESB Adapter) allows AppServer application services to be accessed as services (OpenEdge services) on the Sonic ESB (Sonic Enterprise Service Bus) using a similar interface to that provided by the WSA to access AppServer application services as OpenEdge Web services. However, in addition to allowing AppServer application services to be accessed as Web services, the Sonic ESB allows them to participate in the same client-service and inter-service interactions as any service hosted on the ESB.

All security for the JMS infrastructure accessed by the SonicMQ Adapter as well as for the ESB infrastructure in which the Sonic ESB Adapter participates is provided by either SonicMQ or the Sonic ESB. For more information, see the Sonic documentation for these products.

However, OpenEdge does support data privacy using SSL, as follows:

- Over connections between the AVM and the OpenEdge Adapter for SonicMQ BrokerConnect. SonicMQ BrokerConnect provides messaging services both for Internet connections for ABL clients indirectly through the AppServer Internet Adapter to the BrokerConnect and for intranet connections for ABL clients directly to the BrokerConnect. For Internet access to the SonicMQ BrokerConnect, OpenEdge provides the following options to maximize compatibility with firewalls:
  - TCP/IP port management
  - Network Address Translation (NAT) and distributed configurations
  - NameServer client port range value settings
  - Direct connection to the SonicMQ BrokerConnect without a NameServer to avoid UDP conflicts with firewalls
- Over intranet connections between the OpenEdge services managed by the OpenEdge Adapter for Sonic ESB and the AppServer.

For more information on SSL in OpenEdge, see Application network security using SSL on page 44. For more information on OpenEdge integration services and their architectures, see *OpenEdge Getting Started: Application and Integration Services*.

Progress Dynamics security features

Progress Dynamics is a Repository-based application development and deployment environment that takes advantage of ABLs dynamic user interface, data, and execution invocation elements to provide a methodological approach to rapid application development, deployment, and maintenance. Dynamics stores all of the definitions for these dynamic application elements in an integrated Repository built using the OpenEdge RDBMS.

Progress Dynamics also supports a rich set of security options to authenticate user and group access to many of the application elements that it defines. Configuration of these options results in a set of security structures stored in the Repository along with the application element definitions that these structures secure. Many of the security features in Dynamics facilitate the creation and maintenance of these security structures in the Repository.

Working with the Progress Dynamics security features includes:

- Setting basic security options on page 43

- Defining Progress Dynamics application security options on page 43
- Using the Progress Dynamics security managers on page 44
- Understanding the Repository security structures on page 44

For basic information on Progress Dynamics, see *OpenEdge Development: Progress Dynamics Getting Started*.

Setting basic security options

The basic security options in Progress Dynamics include:

- Setting global options:
  - Choosing between a grant and revoke security model
  - Enabling/disabling the security model
  - Setting login and password options
  - Setting session time-out limits
  - Setting controls on accessing programs in multiple sessions and according to a single user ID
- Setting up user authentication
- Defining *login companies* (a mechanism to categorize groups of users)
- Defining security groups
- Creating and maintaining users, including:
  - Assigning passwords
  - Setting up *profile users* (users whose profiles can be used to define other users)
  - Linking users and security groups

For more information on these basic security options, see *OpenEdge Development: Progress Dynamics Administration*.

Defining Progress Dynamics application security options

Application security options include using the Dynamics security documentation and the Security Control, a tool to manage all levels of application security, including:

- Maintenance for creating and modifying:
  - Login companies
  - Security groups
  - Users
  - Security structures that reference related Dynamics objects, such as fields, menus, and data ranges
- Security allocations that associate users with security structures in order to grant or revoke access rights, according to the selected security model
- Security inquiry for analyzing the current security model, its structures, and relationships

- Security processing for creating security groups that match the security definition of a user

For more information on these application security options, see *OpenEdge Development: Progress Dynamics Basic Development*.

Using the Progress Dynamics security managers

Progress Dynamics includes a set of special-purpose Dynamics objects called managers that provide internal services to Progress Dynamics applications. Among these include two managers that support Dynamics security:

- Security manager
- Profile manager

For information on using the security and profile managers, see *OpenEdge Development: Progress Dynamics Advanced Development*. For information on the security and profile manager APIs, see *OpenEdge Development: Progress Dynamics Managers API Reference*.

Understanding the Repository security structures

A key to understanding the Progress Dynamics security structures is how they are set up in the Dynamics Repository. For more information, see *OpenEdge Development: Progress Dynamics Repository Reference*.

Application network security using SSL

OpenEdge supports data privacy and client/server authentication over connections between OpenEdge clients, servers, and middleware using the Secure Sockets Layer (SSL). This SSL support works at two levels of networking:

- **Secure Internet access** — Using OpenEdge middleware and third-party Web servers or Java Servlet Engines (JSEs) to secure Internet connections between an OpenEdge client and server component using HTTPS (HTTP tunneled through SSL)
- **Secure intranet access** — Securing direct connections between OpenEdge client, middleware, and server components using an OpenEdge implementation of SSL (TCP/IP tunneled through SSL)

Depending on the OpenEdge components involved, Internet and intranet SSL can be used alone or together to provide combinations of security options for certain OpenEdge client and server connections.

As you might know or have surmised from the basic functionality provided by this security option (data privacy and client/server authentication), SSL fundamentally relies on various forms of data cryptography that are specially organized to provide this connection security. To help understand the scope and impact of using SSL with OpenEdge, this manual describes this security option at several levels starting with the technical foundations and basic terminology required to understand and describe SSL itself as well as its role in OpenEdge.

For information on:

- The foundation technologies and terminology for SSL, see Public-Key Infrastructure (PKI) on page 69
- The features and functions of SSL, see Public-Key Infrastructure (PKI) on page 69
- How OpenEdge uses SSL to secure its client, server, and middleware components, and how to configure these components to use SSL, see Secure Sockets Layer (SSL) on page 77
- How to prepare the OpenEdge platform environment to use SSL, see the sections on managing OpenEdge key and certificate stores in *OpenEdge Getting Started: Installation and Configuration*

Caution: SSL in OpenEdge can provide significant, though not perfect, security for Internet and intranet connections. Also be aware that any use of SSL can have significant negative impact on application performance. So, be sure that you need the security that SSL provides before designing and building it into your applications.

Auditing security

Auditing is a core business service (see OpenEdge Core Business Services on page 23) that can affect broad areas of an OpenEdge application defined by various classes of events, including database, internal system, and application events. Auditing security controls access to auditing policies and all the auditing data generated and contained within an audit-enabled OpenEdge database. It controls who can set auditing policies and who can access the auditing data generated from those policies, and how they can access that data.

Several mechanisms contribute to auditing security in the following OpenEdge components:

- **ABL** — Elements of the language that support user authentication and authorization of auditing and database access privileges. For more information on user authentication and authorization in ABL, see Core user authentication and authorization on page 45.
- **OpenEdge SQL** — Elements of the language that support user authentication and authorization of auditing and database access privileges. For more information on user authentication and authorization in SQL, see Core user authentication and authorization on page 45.
- **OpenEdge RDBMS** — Authentication and authorization of users to access auditing data through various database administration and auditing tools. Also, the sealing of audit data against tampering.

Auditing security affects both application development and administration. For more information on OpenEdge auditing security, see Part III, Auditing.

Core user authentication and authorization

OpenEdge provides authentication and authorization features to broad areas of an OpenEdge application through elements of ABL and OpenEdge SQL. These features include mechanisms for:

- Configuring OpenEdge security domains in an OpenEdge RDBMS on page 45
- Creating user IDs in the OpenEdge RDBMS _User table on page 46
- Authenticating and authorizing a user identity on page 46
- Asserting a user identity that is validated using single sign-on (SSO) on page 47

Configuring OpenEdge security domains in an OpenEdge RDBMS

An OpenEdge security domain identifies an authentication system used to validate user identities that authenticate in that domain, and also specifies common attributes associated with all successfully authenticated users in the domain. Each domain is configured in an OpenEdge RDBMS, and is specified by a domain name that is also part of the identity of every user who successfully authenticates to an OpenEdge database client. OpenEdge supports some authentication systems, including user-defined authentication systems configured to allow OpenEdge to authenticate a users identity. OpenEdge supports other authentication systems, including user-defined authentication systems configured for single sign-on (SSO) only, that require an ABL application to authenticate the users identity.

When a user provides their user ID to OpenEdge or an ABL application for user authentication, it must either implicitly or explicitly include the name of an OpenEdge security domain that tells OpenEdge or the application what authentication system to use. For more information on OpenEdge security domains and their configuration, see *OpenEdge Getting Started: Identity Management*.

Creating user IDs in the OpenEdge RDBMS _User table

You can create user IDs in the OpenEdge RDBMS `_User` table for authentication in both ABL and OpenEdge SQL. You can use one of the following mechanisms to create user IDs in the OpenEdge RDBMS:

- OpenEdge Database Administration. For more information on creating user IDs with the Database Administration tool, see *OpenEdge Data Management: Database Administration*.
- Direct ABL access to system tables that store user information. For more information on accessing OpenEdge database tables from ABL, see *OpenEdge Development: Programming Interfaces*.
- The OpenEdge SQL CREATE and ALTER USER statements. For more information on creating user IDs in SQL, see *OpenEdge Data Management: SQL Development*.

Authenticating and authorizing a user identity

From ABL clients, SQL clients, or database utilities, you can authenticate a user ID found either in the OpenEdge RDBMS `_User` table, or in other user accounts supported by OpenEdge built-in or user-defined authentication systems, using the following features:

- **User ID `(-U)` and Password `(-P)` startup parameters** — Authenticates the user ID specified on the ABL command line or in the ABL `CONNECT` statement to authentication systems that allow OpenEdge to perform the user authentication, including the OpenEdge `_User` table and local operating system accounts. SQL clients and database command-line utilities have similar mechanisms for authenticating a user ID and password. User-defined authentication systems that support ABL application-performed user authentication cannot authenticate users from the command line or `CONNECT` statement. However, user-defined authentication systems that support OpenEdge-performed authentication can authenticate users through the `CONNECT` statement, but not from the command line.
- **`SETUSERID` ABL function** — Authenticates a specified user ID and password only to the OpenEdge database `_User` table. If authenticated, this function also sets the user identity for a database connection, with the user ID returned by the `USERID` function.
- **`SET-DB-CLIENT` ABL function** — Authenticates a user identity stored in a client-principal object to authentication systems that allow OpenEdge to perform the user authentication, including the OpenEdge `_User` table, local operating system accounts, and any other user accounts managed by a user-defined authentication system that is enabled for user authentication. If authenticated, this function also sets the user identity for one or more database connections in a session, with the user ID returned by the `USERID` function. This function also supports the validation and setting of a previously authenticated database connection identity using single sign-on (see Asserting a user identity that is validated using single sign-on (SSO) on page 47).
- **`SET-CLIENT( )` ABL method on the `SECURITY-POLICY` handle** — Authenticates a user identity stored in a client-principal object to authentication systems that allow OpenEdge to perform the user authentication, including the OpenEdge `_User` table, local operating system accounts, and any other user accounts managed by a user-defined authentication system that is enabled for user authentication. If authenticated, this method also sets the user identity for the ABL session, and optionally as the identity for every available database connection in the session. This method also supports the validation and setting of a previously authenticated database connection or ABL session identity using single sign-on (see Asserting a user identity that is validated using single sign-on (SSO) on page 47.

In ABL, you can authorize an authenticated user ID for permissions specified by application resources and OpenEdge RDBMS tables and fields using the following ABL features:

- **`Can-*` permissions** — Specify these permissions based on user ID pattern lists through OpenEdge Data Administration to authorize the connected user ID for table and field operations at ABL compile time and (optionally) at run time.
- **`CAN-DO` function** — Use this function to authorize one or more user IDs against a list of permissions based on user ID pattern lists that specify whether a user can access certain procedures or features.

These two authorization mechanisms use exactly the same mechanism to evaluate user access permissions.

For more information on authentication and authorization in ABL, see *OpenEdge Getting Started: Identity Management*, and in the sections on application security in *OpenEdge Development: Programming Interfaces* and *OpenEdge Deployment: Managing ABL Applications*.

In OpenEdge SQL, users are automatically authenticated and authorized against user IDs and SQL privileges specified in the OpenEdge RDBMS. SQL privileges are automatically available to users once they have been granted by the SQL DBA or another SQL user who has the privilege to grant them using the SQL `GRANT` statement. Users can be blocked from privileges by another SQL user who has the privilege to revoke them using the SQL `REVOKE` statement.

Note: OpenEdge SQL also allows DBAs to create user definitions—in the database `_User` table—that are only visible to and used by the OpenEdge SQL, and are transparent to OpenEdge Advanced Business Language (ABL).

For more information on authentication and authorization in OpenEdge SQL, see *OpenEdge Getting Started: Identity Management* and *OpenEdge Data Management: SQL Development*.

Asserting a user identity that is validated using single sign-on (SSO)

In ABL, you can assert a user identity that has already been authenticated by any trusted authentication system using the following elements:

- A sealed client-principal object
- `SET-CLIENT( )` ABL method on the `SECURITY-POLICY` system handle
- `SET-DB-CLIENT` ABL function
- An ABL session or local database domain registry that has been loaded from the domain configurations in an OpenEdge database or built by an ABL application

Either `SET-CLIENT( )` or `SET-DB-CLIENT` take a sealed client-principal object and validate its user identity against a trusted domain registry, and if successful, sets the identity to an ABL session, a database connection, or an entire application with many connected ABL sessions and database connections, depending on method or function and how you call it. SSO validates a user identity that might have been authenticated at another location, but in exactly the same security domain. This user identity can be stored and passed around an application independent of the ABL session or database connection for which it was originally authenticated. For more information on managing identities for both session and database access, see *OpenEdge Getting Started: Identity Management*. For more information on asserting identities as a trusted user from ABL, see the sections on application security in *OpenEdge Development: Programming Interfaces*.

SSL Security

The SSL security in OpenEdge includes:

- Support for SSLv3, TLSv1.0, TLSv1.1, TLSv1.2, and TLSv1.3 protocols that also secure applications against the Padding Oracle On Downgraded Legacy Encryption (POODLE) vulnerability. For a list of all supported ciphers for each protocol, see Supported protocols, ciphers, and certificates for Progress OpenEdge clients and servers.
- Default support for digital certificate validation if signatures use SHA1 and SHA2 (256 or 512) message digest algorithms

Note: OpenEdge 12.0 has both TLS 1.3 and TLS 1.2 as default protocols. This is due to the fact that Java 8 does not support TLS 1.3.

POODLE is a vulnerability which can affect secure communications making use of the Secure Socket Layer (SSL) 3.0 or TLS 1.0. However, newer releases of secure communication protocols, including Transport Layer Security (TLS) 1.1 and TLS 1.2, are increasingly less susceptible. An OpenEdge application is vulnerable to POODLE attack if it establishes secure communications over SSL 3.0.

OpenEdge now communicates over TLS 1.2/1.3 by default that makes it safe against this vulnerability. However, you can change the defaults if you are using an OpenEdge client prior to 11.6 to work with 11.6 OpenEdge server. To achieve this, you must downgrade the 11.6 OpenEdge servers to use TLS 1.0 and AES128-SHA and change the default server certificate from SHA256 to SHA1.

POODLE allows an attacker to decrypt cipher text using a padding oracle side-channel attack. Applications that use cipher-block chaining with the SSL 3.0 communication protocol are vulnerable to POODLE attacks. The SSL 3.0 protocol does not adequately check the padding bytes that are sent with encrypted messages. These padding bytes can be replaced by the attacker. So, an attacker can exploit this vulnerability to decrypt and extract information from inside an encrypted transaction.

Note: For more information on the encryption flaws that POODLE takes advantage of, see https://www.openssl.org/~bodo/ssl-poodle.pdf.

Changing the cryptographic protocol, ciphers, and certificates

Progress OpenEdge supports TLS 1.2/1.3 as the default SSL protocol, and SHA 256 and SHA 384 as the default signed server certificates. The table in Supported protocols, ciphers, and certificates for Progress OpenEdge clients and servers on page 49 lists the default ciphers. If you use an OpenEdge client or server prior to 11.6, you can update the default protocol or cipher to one of the supported protocols or ciphers as listed in the tables below.

For example, if you use a client prior to 11.6, it uses TLS 1.0 as the default protocol and AES128-SHA as the default cipher. An OpenEdge 11.6 server uses TLS 1.2 as the default protocol and and the default ciphers are also different as listed in the table. Thus, for compatibility between these two versions, you must downgrade OpenEdge11.6 server to use TLS 1.0 and AES128-SHA. Also the 11.6 default server certificate must be changed from SHA256 to SHA1. See Changing the default protocols and ciphers on page 52 for more details on changing the default.

SSL/TLS libraries supported in OpenEdge

OpenEdge uses several third-party SSL/TLS libraries. The following table identifies OpenEdge components by their SSL/TLS library.

Table 1: OpenEdge components by SSL/TLS library

SSL/TLS library	OpenEdge component
OpenSSL	• OpenEdge ABL applications (Character and GUI) • OpenEdge RDBMS • ODBC
JSSE	• OpenEdge Management • Progress Application Server (PAS) • Java OpenClients • JDBC
Microsoft SSL	.NET OpenClient

Note: In OpenEdge 12.0, the OpenSSL library is upgraded to 1.1.1.

Supported protocols, ciphers, and certificates for Progress OpenEdge clients and servers

The default protocols are `TLS 1.2` and `TLS 1.3`. In OpenEdge 12.0, an ABL socket client or server supports `TLS 1.3` and can communicate with any server or client that supports `TLS 1.3`.

The supported `TLS 1.3` ciphers for client and servers are:

Ciphers for clients	Ciphers for servers
`TLS_AES_256_GCM_SHA384`	`TLS_AES_256_GCM_SHA384`
`TLS_CHACHA20_POLY1305_SHA256`	`TLS_CHACHA20_POLY1305_SHA256`
`TLS_AES_128_GCM_SHA256`	`TLS_AES_128_GCM_SHA256`

The default SSLv3, TLSv1.0, TLSv1.1, and TLSv1.2 ciphers for clients and servers are:

Ciphers for clients	Ciphers for servers
AES128-SHA256	AES128-SHA256
DHE-RSA-AES128-SHA256	DHE-RSA-AES128-SHA256
AES128-GCM-SHA256	AES128-GCM-SHA256
DHE-RSA-AES128-GCM-SHA256	DHE-RSA-AES128-GCM-SHA256
ADH-AES128-SHA256	AES256-SHA256
ADH-AES256-SHA256	DHE-RSA-AES256-SHA256
AES256-SHA256	ADH-AES128-SHA256
DHE-RSA-AES256-SHA256	ADH-AES256-SHA256
AES256-GCM-SHA384	
DHE-RSA-AES256-GCM-SHA384	
ADH-AES256-GCM-SHA384	

When you install OpenEdge, the default protocols are enabled. You can change the default to the other supported protocols, ciphers, or certificates. The following table lists the compatibility matrix between protocols, ciphers, and certificates.

Note: All the cryptographic protocols and ciphers are supported by default. You can use either the short name or the long name of the ciphers.

Protocols	Ciphers	Certificates
TLSv1.3	TLS_AES_256_GCM_SHA384	The default server certificate must be signed with SHA384. Do the following for the server certificate: 1. Take a backup of $DLC/keys/default_server.pem (that is signed with SHA256) 2. Rename $DLC/keys/test_server_SHA384.pem to $DLC/keys/default_server.pem.
TLSv1.3	TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256	The server certificates are signed with SHA256 ($DLC/keys/default_server.pem). If you use your own certificates, make sure they are signed with SHA256.

Protocols	Ciphers	Certificates
TLSv1.2	AES128-SHA256 DHE-RSA-AES128-SHA256 AES128-GCM-SHA256 DHE-RSA-AES128-GCM-SHA256 ADH-AES128-SHA256 ADH-AES256-SHA256 AES256-SHA256 DHE-RSA-AES256-SHA256	Default server certificate is signed with SHA256 ($DLC/keys/default_server.pem). If you use your own certificates, make sure they are signed with SHA256.
TLSv1.2	AES256-GCM-SHA384 DHE-RSA-AES256-GCM-SHA384 ADH-AES256-GCM-SHA384	The default server certificate must be signed with SHA384. Do the following for server certificates: 1. Take a backup of $DLC/keys/default_server.pem (that is signed with SHA256) 2. Rename $DLC/keys/test_server_SHA384.pem to $DLC/keys/default_server.pem.
TLSv1.1 TLSv1.0 SSLv3	AES128-SHA RC4-SHA DES-CBC3-SHA DES-CBC-SHA EXP-DES-CBC-SHA	The server certificates must be signed with SHA1. Do the following: 1. Take a backup of $DLC/keys/default_server.pem (that is singed with SHA256) 2. Rename $DLC/keys/test_server_SHA.pem file to $DLC/keys/default_server.pem.

The list of ciphers depend upon the ciphers supported by the vendor that you are using, for example, JSSE.

Here are the ciphers supported for JSSE:

- AES128-SHA256
- DHE-RSA-AES128-SHA256

Note: The list of ciphers are updated across releases, see release-specific OpenEdge documentation for supported ciphers.

Note:

- OpenEdge 12.0 does not support the RSA library for `SSL/TLS` connections.
- OpenEdge 12.0 and OpenEdge 11.7 do not support `AES128-SHA` cipher with `SSLv3` protocol on the AIX platform.
- If you update a protocol, the supported ciphers are not updated automatically, you must update to one of the supported ciphers (as listed in the table above) for the changed protocol manually.
- If you use any `AES256-*` ciphers or a server certificate on any platform with more than 2048 keysize, do the following:
  1. Take a backup of the `local_policy.jar` file in `$DLC/jdk/jre/lib/security` and `$DLC/jre/lib/security` locations.
  2. Copy the `local_policy.jar` file in the `$DLC/java/ext/` location to `$DLC/jdk/jre/lib/security` and `$DLC/jre/lib/security` locations.
  3. Restart the admin server and other related components.
- OpenEdge supports `TLS 1.0` for .NET Framework 4.0 and `TLS 1.2/1.3` for .NET Framework 4.5.
- If you use Apache Web Server, see the Apache documentation for the security considerations.

Changing the default protocols and ciphers

Progress OpenEdge provides a set of configuration properties for you to change the default cryptographic protocols and ciphers across all OpenEdge client and server components.

Changing the default protocols and ciphers for Progress OpenEdge clients

You can set the protocols and the ciphers for the following client components:

- **ABL Client**:
  - To set the protocol and cipher for all ABL client connections, export the following environment variables in the startup script of the client application:

Property	Description
`PSC_SSLCLIENT_PROTOCOLS`	Accepts the cryptographic protocol that is set for the client
`PSC_SSLCLIENT_CIPHERS`	Accepts the cryptographic cipher that is set for the client

    Once you export the variables, invoke a Progress OpenEdge client executable and start a session for the client application to use the set cryptographic protocols and ciphers.
  - To set the protocol and cipher for single client connection, you can set the following parameters in the connection-parameters argument of the client's `CONNECT()` method:

Connection parameter	Description
`-sslprotocols` *SSL-protocol-names*	Accepts the cryptographic protocol that is set for the client
`-sslciphers` *SSL-protocol-ciphers*	Accepts the valid cryptographic ciphers that is set for the client

Note: You can use ABL clients to create connections with different server components. If you do not set the `-sslprotocols` and `-sslciphers` connection parameters using the client's `CONNECT()` method, the protocol and cipher values are set using the `PSC_SSLCLIENT_PROTOCOLS` and `PSC_SSLCLIENT_CIPHERS` environment variables in the `ubroker.properties` file (available at *<OpenEdge-install-directory>*`/properties`).

- **Java Open Client**: In the client's `Connection` object, you can set the following properties using the `setStringProperty` method of the `RunTimeProperties` class:

Property	Description
`PROGRESS.Session.sslProtocols`	Accepts the cryptographic protocol that is set for the client
`PROGRESS.Session.sslCiphers`	Accepts the valid cryptographic cipher that is set for the client

  You can also set these properties as Java system properties.

- **.NET Open Client**: In the client's `app.config` file, you can set the `Progress.Open4GL.RunTimeProperties.SSLProtocols` runtime property with the cryptographic protocol. The cryptographic cipher is automatically set using the .NET framework.

- **REST Adapter**: In the `runtime.props` file (available at `$CATALINA_HOME/webapps/application_name/WEB-INF/adapters`), set the following properties:

Property	Description
`sslProtocols`	Accepts the cryptographic protocol that is set for the REST application
`sslCiphers`	Accepts the valid cryptographic cipher that is set for the REST application

Changing the default protocols and ciphers for Progress OpenEdge servers

You can set the protocol and the cipher for the following server components:

- **OpenEdge Database Server**: In the startup script of the server application, export the following environment variables.

Property	Description
PSC_SSLSERVER_PROTOCOLS	Accepts the cryptographic protocol that is set for the database server instance
PSC_SSLSERVER_CIPHERS	Accepts the valid cryptographic cipher that is set for the database server instance.

Once you export the variables, invoke a database server executable and start a session for the client application to use the set protocols and ciphers.

- **OpenEdge RDBMS**: In the `ubroker.properties` file (available at *<OpenEdge-install-directory>*`/properties`), you can set the following environment variables:

Property	Description
PSC_SSLSERVER_PROTOCOLS	Accepts the cryptographic protocol that is set for the instance
PSC_SSLSERVER_CIPHERS	Accepts the valid cryptographic cipher that is set for the instance.

- **MS SQL Server**: In the `ubroker.properties` file (available at *<OpenEdge-install-directory>*`/properties`), you can set the following environment variables:

Property	Description
PSC_SSLSERVER_PROTOCOLS	Accepts the cryptographic protocol that is set for the instance
PSC_SSLSERVER_CIPHERS	Accepts the valid cryptographic cipher that is set for the instance.

- **OpenEdge Management and OpenEdge Explorer**:
  - **WebServer**: In the `fathom.properties` file (available at `$OEMgmt/conf`), you can set the following environment variables:

Property	Description
HttpsEnabled	Enables you to change the cryptographic protocols and ciphers for secure communication with a WebServer. If you enable SSL for the WebServer in OpenEdge Management and OpenEdge Explorer, this property is set to `true`.

Property	Description
SSLEnabledProtocols	If you want to change the default cryptographic protocol for the WebServer, enter this property in the fathom.properties file. The property accepts a comma-separated list of valid cryptographic protocols that are set for secure communication.
SSLEnabledCipherSuites	If you want to change the default cryptographic ciphers for the WebServer, enter this property in the fathom.properties file. The property accepts a comma-separated list of valid cryptographic ciphers that are set for secure communication.

Additionally, in the fathom.init.params file (available at $DLC), you can set the the property ssl.KeyManagerFactory.algorithm=IbmX509.

Note: You can set the ssl.KeyManagerFactory.algorithm=IbmX509 property only on AIX, AIX (64-bit), and LinuxPPC systems.

Note:

In OpenEdge 11.3.3, OpenEdge Management is configured to use Jetty Web server version 7.6.3. Even if you successfully enable secure communications using the Webserver, you may see the following warning message about the secure connection closing occuring repeatedly in the AdminServer log file:

```
...
Invoking E-mail action test. Action: Default_Clear_Action, From: admin@domain.com,
 To: user@domain.com (9614)
2015-06-19 18:36:27.088:WARN:oeji.nio:javax.net.ssl.SSLException:
Inbound closed before receiving peer's close_notify: possible truncation attack?
Invoking E-mail action test. Action: Default_Clear_Action, From: admin@domain.com,
 To: user@domain.com (9614)
2015-06-19 18:36:27.518:WARN:oeji.nio:javax.net.ssl.SSLException:
Inbound closed before receiving peer's close_notify: possible truncation attack?
...
```

This behavior is expected and these messages can be ignored.

- **Email alerts configuration**: In the fathom.properties file (available at $OEMgmt/conf), you can set the following environment variables:

Property	Description
SmtpSSLEnabledProtocols	If you want to change the default cryptographic protocol for the email alerts configuration, enter this property in the fathom.properties file. The property accepts a comma-separated list of valid cryptographic protocols that are set for secure communication.
SmtpSSLEnabledCipherSuites	If you want to change the default cryptographic ciphers for the email alerts configuration, enter this property in the fathom.properties file. The property accepts a comma-separated list of valid cryptographic ciphers that are set for secure communication.

Adding Server Name Indication (SNI)

Server Name Indication (SNI) is an extension to the TLS computer networking protocol. It allows the client to add the hostname that it attempts to connect to during the handshake as a part of the TLS negotiation. It enables the server to select the required domain name and present the certificate with the correct name. It also allows the server to present multiple certificates on the same IP address and TCP port number and thus allow multiple secure (HTTPS) websites (or any other Service over TLS) to be served off the same IP address without requiring all those sites to use the same certificate.

OpenEdge supports SNI only for ABL Client and .NET Open Client.

Set the following parameters in the connection-parameters argument of the ABL Client's CONNECT() method:

-servername

Specifies the server name in SSL connection initialization for client that it sends to the server as part of the TLS negotiation.

Here is an example of the ABL Client CONNECT() method:

```
IF happsrv1:CONNECT("-URL https://localhost:8443/ -servername www.example.com") THEN
```

Note: If your ABL client uses WSDL and SOAP connections where WSDL and SOAP services are configured in two machines (using two different Tomcat servers), use the -wsdlServername and -soapServername respectively. For example,

```
 IF happsrv1:CONNECT("-URL https://localhost:8443/ -wsdlServername www.server1.com
-soapServername www.server2.com") THEN.
```

You can set SNI for a .NET client by using the following property:

```
Progress.Open4GL.RunTimeProperties.ServerName = "servername";
```

`servername`

Specifies the server name in SSL connection initialization for client that it sends to the server as part of the TLS negotiation.

3

Cryptography

Cryptography (commonly referred to as *crypto*) is, in part, the process of scrambling data that is both generally accessible and readable into a form that is both generally inaccessible and unreadable; it is also the process of unscrambling data that has previously been scrambled so it is generally accessible and readable, again. The process of scrambling readable data into an unreadable form is known as *encryption*. The process of unscrambling previously scrambled data so that it is, again, readable is known as *decryption*.

There are many techniques for scrambling and unscrambling data, but they largely derive from a few basic types. This chapter describes the basic types of cryptography and explains how they are supported in OpenEdge.

For details, see the following topics:

- Basic cryptography
- Symmetric (secret) key cryptography
- Password-based encryption (PBE)
- Asymmetric (public) key cryptography
- Message digests
- Managing cryptographic data

Basic cryptography

All basic security services in an enterprise depend on how securely and practically various kinds of data can be encrypted and decrypted—at the right time, in the right place, and by the right people. Cryptography represents the foundation technology upon which all security is built.

Algorithms

Security systems rely on several types of cryptography in order to provide a practical and manageable data security infrastructure. The differences between them depend on the type of *algorithm* (methodology, also known as a *cipher*) used to perform encryption or decryption. The differences in algorithms reflect different trade-offs in complexity (level of security), speed, and practical usability for the task at hand. These same differences exist to a lesser extent between variations of the same type of algorithm. These variations depend on different sizes and types of secret data values (keys) that are input to the algorithm, and—for some types of algorithms—different modes of operation. Also, some algorithms (known as *hashing* algorithms) perform a type of one-way encryption (encryption with no decryption).

Level of security (*strength*) and speed are perhaps the two most important characteristics of a cryptographic algorithm. Security systems often rely on algorithms of different strengths and speeds for different aspects of security.

Algorithms are typically identified by a meaningful acronym, such as DES (Data Encryption Standard) or RSA (the initials of its three inventors). This acronym distinguishes the type of algorithm within the larger type of cryptography that it represents, and is also unique among different types of cryptography. (For example, while DES is a symmetric-key algorithm, there is no asymmetric-key algorithm also identified as DES.)

Keys

All cryptographic algorithms rely on one or more keys. A *key* is a discrete piece of data, sometimes in the form of a password, that might both "lock" and "unlock" the encryption and decryption process, depending on the type of algorithm. The types and variations of algorithm also determine the sizes of the keys that are used with an algorithm (size specified in bits, for example, 256). Generally, for a given algorithm type, the larger the key, the stronger it makes the algorithm; also for a given algorithm type, the larger the key, the slower it makes the algorithm.

Symmetric (secret) key cryptography

This is the oldest and best understood type of cryptography. Its algorithms offer a wide range of security and speed, depending on the algorithm type and variation. *Symmetric-key cryptography* generally relies on a single, shared key to perform both encryption and decryption on the same piece of data. Thus, the secrecy of the key value is of critical importance to the security of the data. As a result, this type of cryptography is sometimes known as *secret-key cryptography*. For more information on one mechanism for maintaining the secrecy of symmetric keys, see Asymmetric (public) key cryptography on page 64.

Encryption/decryption algorithms

The encryption/decryption algorithms vary in complexity and speed and rely on iterative applications of Exclusive OR (XOR) on various combinations of key and data. This range of complexity and speed makes symmetric key algorithms a common choice for providing data privacy and authentication services. Three common algorithms, used by OpenEdge, are:

- **DES** — Data Encryption Standard
- **DES3** — Triple DES
- **AES** — Advanced Encryption Standard

The AES cipher is the current U.S. government standard for all software, and is recognized worldwide.

The DES ciphers are primarily supported for PBE standard that provides the option of generating an encryption key based on a passphrase.

Encryption/decryption modes

Encryption algorithms can have different *modes of operation (modes)*. A *mode* is a kind of companion algorithm that tailors the symmetric-key algorithm for different applications, such as confidentiality or authentication. Two modes for the algorithms supported by OpenEdge are:

- **CBC**—Cipher block chaining. CBC is a mode of operation for a block cipher in which a sequence of bits are encrypted as a single unit or block with a cipher key applied to the entire block. Cipher block chaining uses an initialization vector (IV) of a certain length. One of the key characteristics of CBC is that it uses a chaining mechanism that causes the decryption of a block of ciphertext to depend on all the preceding ciphertext blocks. As a result, the entire validity of all preceding blocks is contained in the immediately previous ciphertext block. A single bit error in a ciphertext block affects the decryption of all subsequent blocks. Rearrangement of the order of the ciphertext blocks causes decryption to become corrupted.
- **ECB**—Electronic Code Book. ECB is a mode of operation for a block cipher with the characteristic that each possible block of plaintext has a defined corresponding ciphertext value and vice versa. In other words, the same plaintext value will always result in the same ciphertext value. Electronic Code Book is used when a volume of plaintext is separated into several blocks of data, each of which is then encrypted independently of other blocks.

  ECB is not a good mode to use with small block sizes (for example, smaller than 40 bits) and identical encryption modes. This is because some words and phrases may be reused often enough that repetitive part-blocks of ciphertext can emerge, laying the groundwork for a code book attack where the plain text patterns are fairly obvious. Security may be improved when random pad bits are added to each block. On the other hand, 64-bit or larger blocks should contain enough unique characteristics (entropy) to make a codebook attack unlikely to succeed.

Encryption/decryption key size

Encryption keys vary in size.The larger the key the harder it is to break. The encryption algorithms supported by OpenEdge have the following key sizes:

- 56 (DES)
- 128 (AES)
- 168 (DES3)
- 192 (AES)
- 256 (AES)

The trade-off between the key sizes is that larger keys take longer to encrypt and decrypt.

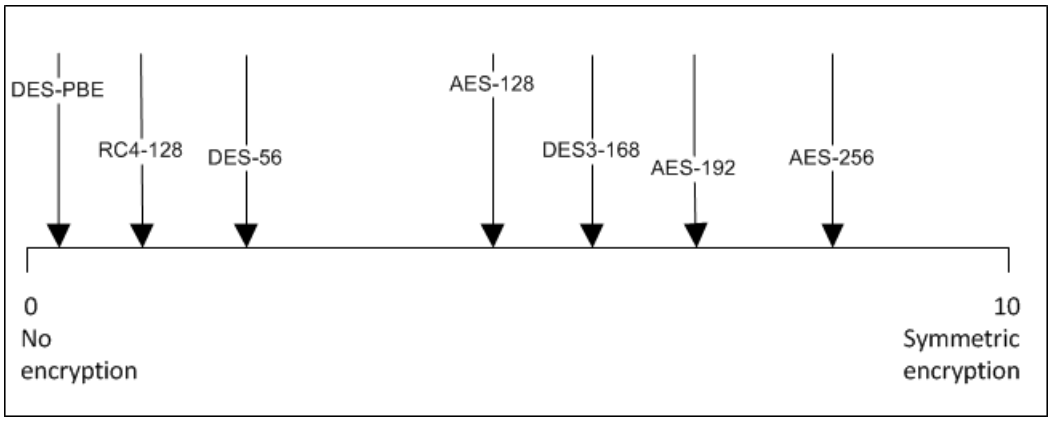
Combine algorithm, mode, and key size

Symmetric-key algorithms are identified by the three components of their supported variations: *`algorithm_mode_keysize`*. So, for example, the symmetric-key algorithm identified by the string `"AES_CBC_128"`, specifies the AES symmetric-key algorithm combined with the CBC mode using a 128-bit key size.

The AES_CBC_128 cipher is the default cipher for OpenEdge RDBMS Transparent Data Encryption. It represents the best balance between the strength of data encryption, its viability in today's encryption market, and performance. The 128 bit key size is chosen because it is the fastest of the AES key sizes.

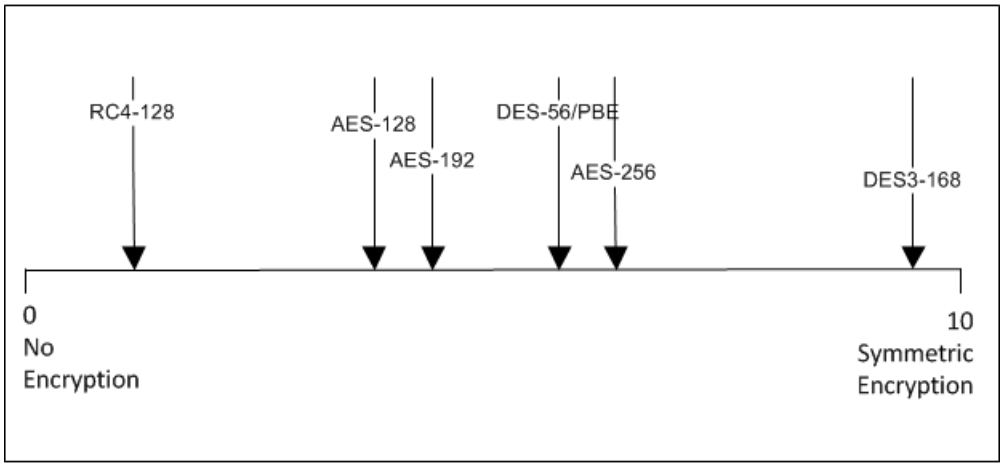
The following figure shows the relative key strengths on a scale of 1 to 10.

Figure 2: Relative key strength



The following figure shows the relative speed of the supported encryption keys on a scale of 1 to 10. Encryption and decryption speed is not absolute; it is highly dependent on system processor speeds.

Figure 3: Relative key encryption and decryption speed



OpenEdge usage

Symmetric-key cryptography is used in OpenEdge in three ways:

- To provide Transparent Data Encryption of database blocks. For more information on Transprent Data Encryption, see OpenEdge Key Store on page 177, or Configuring Transparent Data Encryption policies on page 185.
- To provide data privacy on Secure Sockets Layer (SSL) connections. SSL is an implementation of a Public-Key Infrastructure (PKI). For more information on:
  - **PKI** — See Public-Key Infrastructure (PKI) on page 69
  - **SSL** — See Public-Key Infrastructure (PKI) on page 69
  - **OpenEdge support for SSL** — See Secure Sockets Layer (SSL) on page 77
- To enable ABL applications to provide their own data privacy and integrity functionality, OpenEdge provides symmetric-key cryptography features directly in ABL. In ABL, these features are supported by the:
  - `GENERATE-PBE-KEY` function (see Password-based encryption (PBE) on page 63

- `GENERATE-PBE-SALT` function (see Password-based encryption (PBE) on page 63)
- `GENERATE-RANDOM-KEY` function
- `ENCRYPT` and `DECRYPT` functions
- XCODE utility for encrypting source code. See *OpenEdge Deployment: Managing ABL Applications* for more information
- `SECURITY-POLICY` system handle to provide a consistent cryptography policy for specifying keys and encryption algorithms for the above functions; note that the `XCODE-SESSION-KEY` attribute of this handle allows you to provide session-specific keys for encrypted source code

For more information on the cryptographic features of ABL, see the sections on cryptography in *OpenEdge Development: Programming Interfaces*.

Caution: Progress Software Corporation recommends that you use the cryptographic features of ABL **only** if you have a well-grounded understanding of cryptography and its usage. Use of cryptography without the necessary preparation can result in permanent data loss. In general, cryptography can have significant negative impact on application performance and decrease effective data compression for data stored in a database.

Password-based encryption (PBE)

Password-based encryption (PBE) is a form of symmetric-key generation that transforms an input string (a password) into a binary encryption key using various data-scrambling techniques. Some form of PBE is typically behind the logon mechanism at most points in an enterprise and is also used for some services of a PKI.

PBE algorithms

PBE is typically implemented using standard hashing algorithms, such as the PKCS#5 standard of RFC2898. These algorithms often use a built-in key to encrypt a password and might also allow the addition of an auxiliary key (a *PBE salt*) to strengthen the encryption.

OpenEdge usage

OpenEdge provides PBE features directly in ABL that are especially designed to generate symmetric encryption keys for the ABL symmetric-key encryption and decryption functions. In ABL, these features include the:

- `GENERATE-PBE-KEY` function
- `GENERATE-PBE-SALT` function

For more information, see the sections on cryptography in *OpenEdge Development: Programming Interfaces*.

Caution: Progress Software Corporation recommends that you use the cryptographic features of ABL **only** if you have a well-grounded understanding of cryptography and its usage. Use of cryptography without the necessary preparation can result in permanent data loss. In general, cryptography can have significant negative impact on application performance and decrease effective data compression for data stored in a database.

OpenEdge also uses PBE to secure the generation and storage of private keys and server digital certificates to establish SSL identities for OpenEdge servers.

For more information on:

- **SSL** — See Public-Key Infrastructure (PKI) on page 69
- **OpenEdge support for SSL** — See Secure Sockets Layer (SSL) on page 77
- **Support for managing private keys and server digital certificates in OpenEdge** — See the sections on managing OpenEdge key and certificate stores in *OpenEdge Getting Started: Installation and Configuration*.

Asymmetric (public) key cryptography

Asymmetric-key cryptography (also known as *public-key cryptography*) relies on a *key pair* to perform both encryption and decryption on the same stream of data. As the name implies, public-key cryptography is what makes a PKI possible (see Public-Key Infrastructure (PKI) on page 69), and it is a much more recent development in cryptography than symmetric (or secret) key cryptography. There are variations on this concept, but the most commonly used algorithms rely on one copy of a key that is owned and kept secret by one entity and multiple copies of a common public key possessed by all other entities that want to communicate with the owner of the private key.

Asymmetric keys

The key that must be kept secret is referred to as a *private key*, to distinguish it from the *secret key* of symmetric-key cryptography. This key is owned by and defines the unique identity of a particular entity (call it a *server entity*). The *public key*, as the name implies, is public and is commonly known to any entity (call it a *client entity*) that wants to exchange confidential communications with the owner of the private key.

The nature of the algorithms used with these key pairs is such that any client entity that holds a copy of the public key can engage in confidential communications with the server entity that owns the corresponding private key, and these communications are also confidential from other holders of the same public key. This works because any message encrypted using the private key can only be decrypted using the public key, and any message encrypted using the public key can be decrypted using only the private key. If a server entity maintains the secrecy of its private key (that is, if the server entity is secure), any client entity with the corresponding public key can know the identity of the secure server entity that it communicates with.

In a PKI, to assure confidentiality from other clients that possess the same public key, any client entity maintains data privacy by initiating its own secret-key communications with the server entity that owns the corresponding private key. This is what makes a PKI possible.

With symmetric-key cryptography alone, any entities seeking to engage in confidential communications must find a secure way to share the same secret key. The problems of managing a data security infrastructure with this constraint are formidable. With asymmetric-key cryptography, as used in a PKI, only one server entity needs to own the private (and secret) key. Any client entity can have access to the same public key without compromising data security for any other clients that want to exchange data with this server entity. Thus, the public key requires no security. The essential security for public-key cryptography depends only on the private key, which must remain completely confidential to the server entity that owns it. However, if server security is compromised, there can be no assurance that the server entity who owns the private key is the proper owner of that key. So, security for the server is critical to all clients that try to access it.

Public-key algorithms

Public-key algorithms base their security on the solving of complex mathematical problems that are reliant on large keys for numbers. The nature of the mathematical problems distinguishes the different algorithms.

Two of the more common algorithms include:

- **RSA** — Named for its inventors, Ron Rivest, Adi Shamir, and Len Adleman, this algorithm works on very large numbers that can be factored into large primes

- **Digital Signature Algorithm (DSA)** — A U.S. Government standard, this algorithm works on large numbers that can be used to determine discrete logarithms

The minimum practical key size for both RSA and DSA keys is typically 1024 bits. As the size of the keys implies, public-key algorithms are highly compute-intensive and are much slower than symmetric-key algorithms, but they are also far more secure. This far greater security and the ease of public key distribution are what makes them ideal as the basis for a data security infrastructure. However, their relatively slow speed makes them less than ideal to provide the sole cryptographic mechanism for all services of a PKI.

OpenEdge usage

OpenEdge relies on public-key cryptography in its support for SSL, which is an implementation of a PKI. This includes support for standard HTTPS connections over the Internet and its own implementation of SSL for secure connections over an intranet. For more information on:

- **PKI** — See Public-Key Infrastructure (PKI) on page 69.
- **SSL** — See Public-Key Infrastructure (PKI) on page 69.
- **OpenEdge support for SSL** — See Secure Sockets Layer (SSL) on page 77.

Message digests

A *message digest* is a fixed-length representation of a variable string of data. This fixed-length value is known as a message digest because it contains a potentially much smaller, yet unique, representation of the original message or data. By comparing the message digest generated in one place with the message digest generated in another place using what is supposed to be the same data, you can tell if the data has actually been changed while moving from one place to the other. If the two message digests are identical, the data has not changed. Thus, because of their fixed and relatively small size, message digests are very convenient to verify the integrity of the original data from which they are generated.

Message digest algorithms

Message digest algorithms rely on cryptographic hash functions to generate a unique value that is computed from data and a unique symmetric key. A *cryptographic hash function* inputs data of arbitrary length and produces a unique value of a fixed length. Because message digest algorithms generate a value that is always used in encrypted form (never decrypted), they are sometimes known as *encryption-only algorithms*.

Adding a unique symmetric key that is shared between a sender and receiver in order to compute a message digest value provides confidentiality to ensure that the message digest cannot be easily changed if the data is changed in an unauthorized or other unexpected manner. Both the sender and receiver of the data (including the senders message digest) must share the same key for the receiver to generate an identical message digest.

If some other agent changes the data between the sender and receiver and hashes their own message digest without the correct key, the new message digest, while representing the data, will not be the same as the message digest computed by the receiver using the correct symmetric key. The resulting value generated from the symmetric key (or MAC key) and the message digest is known as a *Message Authentication Code (MAC)* because it can be used to test for unauthorized alteration of both the data and the message digest.

OpenEdge usage

OpenEdge provides message digest features in ABL that you can use to provide your own data integrity tests. The most direct is the `MESSAGE-DIGEST` function that produces a message digest given a hashing algorithm, data, and a hash key. This function supports the following algorithms: SHA-1, SHA-256, SHA-512, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512.

OpenEdge also allows you to specify the hashing algorithm for PBE (see Password-based encryption (PBE) on page 63 using a choice of message digest algorithms.

For more information, see the sections on cryptography in *OpenEdge Development: Programming Interfaces*.

Caution: Progress Software Corporation recommends that you use the cryptographic features of ABL **only** if you have a well-grounded understanding of cryptography and its usage. Use of cryptography without the necessary preparation can result in permanent data loss. In general, cryptography can have significant negative impact on application performance and decrease effective data compression for data stored in a database.

OpenEdge also uses message digests and MAC keys to:

- Provide data privacy for an SSL session. For more information, see Public-Key Infrastructure (PKI) on page 69.
- Support security options for Java Open Clients. For more information, see *OpenEdge Development: Java Open Clients*.
- Provide enhanced source code version detection for WebClient applications. For more information, see the section on compiling the application in *OpenEdge Deployment: WebClient Applications*.
- Seal audit data from unauthorized alteration. For more information, see Auditing in OpenEdge on page 97.

Managing cryptographic data

Cryptographic data requires special attention to manage and transport. Such data includes:

- Source data, typically in clear-text form
- Encrypted or hashed data
- Passwords and key values
- Algorithms
- All platform parameters required to maintain the data, such as code pages for character strings and the byte-endian order for binary values

Objectives

The primary objectives for managing this data include:

- Ensuring that the required level of security is maintained as data is moved and stored
- Ensuring that secured data can always be recovered by those who are authorized to do so

Requirements

The requirements for managing cryptographic data depend upon the:

- Type of cryptography involved
- Media used for storage and transport

A general requirement that applies to all cryptography is to ensure that all keys, once generated, are recoverable.

OpenEdge solutions

For cryptographic data that you work with in ABL and that is used in symmetric-key cryptography, PBE, message digests, and MACs, ABL provides additional tools to manipulate this data for transport, including:

- The `GET-BYTE` and `PUT-BYTE` functions to help maintain byte-endian order
- The `BASE64-ENCODE` and `BASE64-DECODE` functions to convert between binary and character representations of machine-level

For more information on the storage and transport of cryptographic data managed in ABL, see the sections on cryptography in *OpenEdge Development: Programming Interfaces*.

For cryptographic data used in public-key cryptography, including especially keys and digital certificates, OpenEdge provides or supports third-party tools to manage this data. For more information, see Secure Sockets Layer (SSL) on page 77 and the sections on managing OpenEdge key and certificate stores in *OpenEdge Getting Started: Installation and Configuration*.

4

Public-Key Infrastructure (PKI)

A *Public-Key Infrastructure (PKI)* implements a methodology, based on the concepts and technology of public-key cryptography, for supporting data security services throughout an enterprise network. As this definition implies, a PKI supports these data security services as part of a common security infrastructure that is implemented throughout the enterprise.

OpenEdge supports elements of a PKI with its implementation of a Secure Sockets Layer (SSL). This chapter describes the basic PKI concepts and terminology used both to define SSL and to describe the OpenEdge implementation of SSL throughout the OpenEdge documentation set.

Caution: Use of a PKI in any network environment can have significant performance impacts. Be certain that you need the security that a PKI supports before you choose to design and build PKI support into any of your application networking.

For details, see the following topics:

- Core security services in a PKI
- Cryptography in a PKI
- Trust relationships and supporting mechanisms

Core security services in a PKI

The core security services provided by an application, and supported by PKI, underlie and enable all of the other security services that an application can provide. They are also among the most commonly used security services in their own right. These core services provide various security assurances to an enterprise application and they include:

- Authentication on page 70
- Confidentiality on page 70
- Integrity on page 70

For basic information on these security services, see the sections on security in OpenEdge Core Business Services on page 23. A PKI supports all of these services using cryptographic mechanisms that serve as guarantors of the assurances provided by these services (see Cryptography in a PKI on page 71.

The sections that follow describe how these services are used in a PKI.

Authentication

Authentication services for a PKI are supported by asymmetric (or public) key cryptography in order to confirm identity matches. Public-key cryptography relies on the existence of a uniquely matched *key pair*. Each key represents a means to uniquely match an identity defined by the other key. One party has a private key from the key pair that is known only to that party and that identifies its identity uniquely to all other parties. All other parties share a public key from that key pair known to all of them that can uniquely authenticate the party that owns the matching private key. This provides a convenient means to allow many unknown parties to authenticate communications with a party that has a given identity. For more information on the use of public-key cryptography in a PKI, see Cryptography in a PKI on page 71.

PKI authentication services support all other core security services used by a PKI, as well as more derivative security services, such as nonrepudiation, which assures that an author who produces data cannot later deny having produced it.

Confidentiality

Confidentiality services supported by a PKI depend on a combination of cryptographic techniques used to protect the data communications between parties. These services typically rely on public-key cryptography to provide a **public** and ready means for unknown parties to authorize a confidential data exchange with one another, and they rely on another form of cryptography "under the covers" (symmetric-key cryptography) to "hide" and protect the actual data in the communications from access by unauthorized parties.

Note: It is possible for a PKI to support confidentiality services using public-key cryptography alone, but implementations of confidentiality services almost always include a symmetric-key cryptography component for practical reasons. For more information, see Cryptography in a PKI on page 71.

Thus, with a confidentiality service supported by a PKI, one of many clients in the enterprise can authenticate communications with a given server and exchange data with that server with reasonable confidence that the data will not be intercepted and read by any unauthorized agent, including other clients talking confidentially with the same server.

Integrity

Integrity services supported by a PKI use cryptographic techniques to allow parties needing to maintain the integrity of their data transmissions to assure receiving parties that the data they receive is the data that was intended and that it was sent from whom it was intended. In practice, like the data cryptography in confidentiality services (see Confidentiality on page 70), a PKI generally supports symmetric-key cryptographic mechanisms that assure data integrity "under the covers" in a communications environment that is authenticated using public-key cryptography. This data integrity service typically makes itself known only when data is found to have been unexpectedly altered, requiring input from the data consumer in order to handle the corrupted data.

Cryptography in a PKI

As described in previous sections, cryptography underlies all of the core services of a PKI, as well as data security in general. This section defines and describes how cryptography supports a PKI. For basic information on cryptography and cryptographic support in OpenEdge, see Cryptography on page 59.

From the viewpoint of a PKI user, the key is generally the most visible part of cryptography. Depending on the type of key, the secrecy of keys is vitally important. The whole security enterprise depends on secret keys remaining secret to all except those who must have access to them. PKI also works because while certain keys must remain secret (or private), one type of key must be made public.

The "public key" in Public-key Infrastructure indicates the fundamental importance of one particular type of cryptography to a PKI. However, in practice, the mechanism of a PKI typically relies on all of the following types of cryptography:

- Symmetric-key cryptography on page 71
- Message digests and MACs on page 71
- Public-key cryptography on page 71
- Password-based encryption (PBE) on page 72

Symmetric-key cryptography

The foundation of all PKI is public-key (also known as asymmetric-key) cryptography (see Public-key cryptography on page 71), which it uses for the primary authentication services that it provides. While very secure, however, this is a relatively slow cryptographic technique. Therefore, a PKI uses symmetric-key cryptography for its combination of speed and strength to handle the encryption and decryption of actual data after it has already been authenticated. The PKI might generate the symmetric keys randomly using a *Pseudo Random Number Generation function (PRNG),* or it might use password-based encryption (less typical) to generate keys based on a user-supplied password (see Password-based encryption (PBE) on page 72). It then uses asymmetric-key cryptography to exchange and maintain the confidentiality of the symmetric keys used to encrypt and decrypt the data.

Message digests and MACs

A PKI also uses message digests and message authentication code (MAC) algorithms to support data integrity services. These include the integrity of data transmitted from point to point over PKI-authenticated connections and the integrity of data that has been authenticated using digital signatures to authenticate the original sender of the data. For more information on digital signatures, see Trust relationships and supporting mechanisms on page 72.

Public-key cryptography

Typically, a PKI relies on public-key (also known as asymmetric key) cryptography as the visible frontline security mechanism for the infrastructure and uses symmetric-key cryptography invisibly behind the scenes to handle the high-volume and some special-purpose cryptographic tasks. This works because a PKI can use the secure, but relatively slow, public-key cryptography between a client and server entity to exchange and authenticate the relatively small symmetric keys that, in turn, are used to provide the actual privacy and integrity for the bulk of the data exchanged between the same client and server. The PKI generates and uses public-key cryptography to secure all symmetric keys without any intervention from users. The users need only have access to their respective private and public-keys to make use of all the services provided by the PKI-generated symmetric keys.

Common uses for public-key cryptography in a PKI include:

- **Key confidentiality** — In all cases where symmetric cryptography is used, the secret keys involved are protected by the frontline public-key encryption and decryption between communicating entities.
- **Authentication using digital certificates** — The authentication services of a PKI typically rely on digital certificates to communicate the public keys between client and server entities for verifying identity. For more information on digital certificates, see Trust relationships and supporting mechanisms on page 72.
- **Nonrepudiation using digital signatures** — The authentication services of a PKI typically rely on digital signatures both to verify that a certain piece of information originates from a specific entity and to prevent that entity from repudiating (disavowing) their authorship of that information. For more information on digital signatures, see the Trust relationships and supporting mechanisms on page 72.

Password-based encryption (PBE)

In a PKI, the owner of a PKI identity typically specifies a password string for a PBE key generation function to symmetrically encrypt the private key of a public-private key pair. Because the private key is the owners identity and only the owner knows the password to be able to access and use the private key, the private key therefore represents the person who knows that password.

Trust relationships and supporting mechanisms

While a PKI uses public-key cryptography to assure data privacy and integrity of the identities it certifies through public keys, this does not, by itself, provide all of the assurances required to implement a PKI. Public-key cryptography, by itself, does not address the larger questions of how users of a PKI can **trust** that an authenticated entity is, indeed, the entity that has been authenticated. Without this trust, especially in an environment where entity identification is crucial, the effectiveness of the entire PKI is suspect. For more information on the basis for trust in a security system, see the sections on security and trust in OpenEdge Core Business Services on page 23.

PKI trust model

To help ensure trust, a PKI relies on a standard trust model that assigns to a third party the responsibility of establishing a trust relationship between any two communicating entities. The model used by a PKI is a strict hierarchical model. At the top is a publicly (or privately) recognized source (*authority*) that everyone using the PKI recognizes and trusts to validate (*authorize* and *certify*) the identities that are part of the PKI. Under this authority might exist subordinate authorities that rely on the top (*root*) authority as the ultimate source of authorization and certification.

The mechanism that is typically used to convey or validate this authorized identity is the *digital certificate*. The authority that is entrusted with issuing digital certificates for the purpose of authorizing and validating identity is a *Certification Authority* (*CA*, also often referred to as a *Certificate Authority*). Again, CAs can be organized in a hierarchy of authority with the ultimate authority at the top being the *root CA* of that CA hierarchy. The strength of a CA rests entirely on the agreement between the holder of an identity that is authorized by the CA on one side and those who communicate with the holder of that identity on the other side to trust the integrity of the CAs authorization of that identity. Among other requirements, the most important is that this identity must be unique to the identity holder, and all parties involved must trust the CA to guarantee this to the extent possible.

Standards have been developed to define digital certificates, and the most widely accepted standard is the X.509 portion of the ISO and CCITT/ITU-T X.500 suite of standards. However, given that a PKI adheres to such a standard and CAs exist to authorize X.509 digital certificates, the problem still remains for users of the PKI to determine what constitutes a *trusted CA*, a CA that they can trust to authorize digital certificates and assure the identities that they represent. The industry supports reputable public CAs, such as RSA, Thawte, and Verisign. However, there are other public CAs and many more private CAs from which the PKI user must choose.

Digital certificates and certificate store

The most widely accepted digital certificate is the *X.509 public-key certificate*, and it is issued by CAs in two major forms:

- A *server digital certificate* issued to the holder of a private key that authorizes the identity established by the private key for the holder.
- A *root CA digital certificate* issued to clients and servers of a server that they use to authenticate the identity of the server when communications between the client and server begin. This authentication occurs by validating the root CA digital certificate against the server digital certificate.
- A trusted CA/root digital (public-key) certificate can be managed for OpenEdge clients and servers that support SSL connections using a root certificate store located in the `OpenEdge-Install-Dir\certs` directory.
- Each OpenEdge SSL client and server requires the root certificate store entry that contains the public-key certificate from the CA who signed and issued the public-key certificate for the SSL client and server that the client needs to access.

Note: A *CA digital certificate* is a digital certificate used to assert and validate the identity of the CA to anyone who is validating a digital certificate that this CA has issued (such as a server digital certificate). A *root CA digital certificate* is a CA certificate that is at the top of the validation chain in the hierarchy of CAs. So, if the validation process does not trust the root certificate, there is no higher authentication authority to go to and the validation operation must fail.

Digital certificates have a number of properties, and one of the most important for a PKI is its specified *lifetime*, the time in which the digital certificate is valid. When a digital certificates lifetime has expired, it can no longer be used to assert or authenticate a servers identity. For more information on managing digital certificates and certificate stores, see *OpenEdge Getting Started: Installation and Configuration*.

Adding a wildcard character to a public key certificate

You can add a wildcard " * " character in the subject name or common name fields in the public key certificate so that you can define a single common name for multiple domain names. For example, a certificate with "*.bedford.progress.com " common name can be used for user1.bedford.progress.com, user2.bedford.progress.com, or user3.bedford.progress.com.

By default, the client matches the hostname with a wildcard common name using the regEx matching technique. For example, *.progress.com is a valid wildcard common name for user.bedford.progress.com. If you want to validate the hostname as per the RFC 6125 guidelines, set the following environment variable to true:

- `PSC_SSLSTRICT_HOSTNAME_VERIFY=<true>` for Java and ABL clients.
- `Progress.Open4GL.RunTimeProperties.StrictHostVerify=<true>` for .Net clients.

Note: OpenEdge Explorer Management support only strict mode.

Adding Subject Alternative Name (SAN) to a digital certificate

Subject Alternative Name (SAN) is an extension to X.509 that lets you specify additional host names (values) to be protected by a single SSL certificate using a `subjectAltName` field. It allows more than one host to use the same copy of a single certificate. At the server-level, you can create multiple virtual hosts and add these hosts to the `subjectAltName` field of the certificate. You generate a certificate with SAN and the clients can connect to the server using `subjectAltName`. Whenever HTTPS request comes to any of the virtual host, the server uses the same certificate for SSL handshake.

A Wildcard certificate can protect all first-level sub-domains on a domain, such as `*.example.com` but it cannot protect both `www.example.com` and `www.example.org`. Hosting multiple SSL-enabled sites on a single server typically requires a unique IP address per site, a multi-domain (such as SAN) certificate with subject alternative names allows you to protect all these sub domains with a single certificate. However, you can use a Wildcard certificate as part of SAN.

Before you generate the digital certificate using the `pkiutil` utility, open the `pscpki.cnf` file in the %DLC%\keys\policy location and add the `subjectAltName` values as follows under the `x509v3_extensions` section:

```
subjectAltName = @alt_names
```

Add the domain names in the `alt_names` section as follows:

```
[alt_names]
          DNS.1 = <value>
          DNS.2 = <value>
```

For example, let's assume that you want to generate certificate for a host www.default.com and two alternate hosts `www.service1.com` and `www.service2.com`. So, you have three virtual hosts `www.default.com`, `www.service1.com`, and `www.service2.com` and their loop back address are 127.0.0.1, 127.0.0.2, and 127.0.0.3 respectively. Do the following:

1. Open the `hosts.txt` file from the `C:\Windows\System32\drivers\etc` location for Windows or the `/etc/` location for Linux.
2. Add the loop back addresses and the host names.

   Note: If there is a loop back address that is already assigned to host, use another loop back address, for example, if 127.0.0.1 is already assigned, assign 127.0.0.4 to the www.default.com host.

3. Verify if the hosts were added, by pinging each host in the Command prompt. For example, type `ping www.default.com`, if you are able to ping each host successfully, the hosts are added.
4. Create a copy of the `pscpki.cnf` file at the `%DLC%\keys\policy` and name it `myssl.cnf`.
5. Edit the `myssl.cnf` to add the following under the `x509v3_extensions` section:

```
subjectAltName = @alt_names
                              [alt_names]
                              DNS.1 = www.service1.com
                              DNS.2 = www.servuce2.com
```

6. Generate the public and private key pair using the `pkiutil -keysize 2048 -newreq example` command. It creates `example.pk1` and `example.pk10` files in the `%DLC%/keys/requests` location. `example.pk1` serves as your private key and `example.pk10` as the public key serves as a Certificate Signing Request (CSR) to a certificate authority (CA).
7. Verify CSR using the `sslc req -text -in %DLC%\keys\requests\example.pk10 -noout` command.
8. Create a self-signed certificate `.cer` file and convert to `.pem` file using `pkiutil`.
9. Open the `.pem` file and verify the `subjectAltName` field by executing the `Proenv> sslc x509 -in %DLC%\keys\example.pem -text -noout` command. The `X509v3 extensions` section displays the `subjectAltName` as follows:

```
X509v3 extensions:
    X509v3 Subject Alternative Name:
        DNS.1 = www.service1.com
        DNS.2 = www.service2.com
```

Digital signatures

A *digital signature* (often confused with a digital certificate) is a means to electronically sign a piece of data using a private key in a manner analogous to a hand-written signature. A digital signature, like a hand-written signature, authenticates the data as having been signed by the owner of the private key that was used to create the digital signature.

Digital signatures are not specifically defined in the X.509 standard. They are, however, legally recognized by many countries as a mechanism that typically follows X.509 recommendations to employ asymmetric cryptography and a one-way hashing function to generate the unique code that constitutes the signature.

In a PKI, digital signatures are used to provide data integrity and authorship, and thus provide the basis for the nonrepudiation of message exchange in a PKI. While not digital certificates themselves, they are generated from the asymmetric-key information provided by a server digital certificate.

Client and server interactions

A PKI typically secures all client and server interactions established for a given enterprise. Typically, when a client initiates a secure connection to a server, the server uses its server digital certificate to assert its identity. The client then verifies the asserted identity against a root certificate provided by a CA that the client trusts. What happens if the client cannot find a root CA certificate that validates the server identity depends on the PKI, but often the client connection fails with some notification to the user with or without options to proceed.

Once a client validates and completes the connection to the server, the server might generate a symmetric key that it encrypts using asymmetric encryption and shares this key with the client to initiate confidential communications. Many other services can be offered as part of the secure connection between the client and server, including all of the core services of the PKI.

Key and certificate management

One of the most important features of a PKI is how it provides for the management of asymmetric keys and public-key certificates for all entities in the enterprise that the PKI protects.

Server identity management

To establish an identity for a PKI server entity requires that the entity first create a private/public-key pair and store the private key, encrypted, in a secure storage location. The public key, with proof of the owners identity, must be submitted to a CA that validates the owners identity and, if valid, issues a digital certificate that contains the owners public key. The location for storing the servers private key is commonly known as a *key store*.

A key store must allow the owner to manage the servers identity securely, so that the secrecy of the private key is not compromised. At a minimum, each private key (*key store entry*) used to establish an identify in the key store must be individually password-protected.

Client certificate management

For clients, the utilities that support PKI interaction typically provide the functionality for storing and managing CA digital certificates used for authenticating server digital certificates. The location for storing the clients digital certificates is commonly known as a *certificate store*. Each certificate in the certificate store represents a single *certificate store entry*. The certificate store typically allows the client to add, update, examine, remove, and restore any removed certificate store entries. As the content is public, it is not typically password-protected.

Digital certificate life-cycle management

As described earlier, digital certificates have a lifetime during which they are considered valid. When this lifetime expires, the certificate can no longer be used for authentication and must be updated to restore its validity. A certificate can also become invalid from being revoked by a CA. Common reasons for which a CA might revoke a digital certificate include a change in job status or suspicion of a compromised private key.

The CA typically provides a means to revoke digital certificates (*certificate revocation*). This process depends on the mechanism that the CA for each certificate makes available to communicate certificate revocation. Typically, a client that utilizes a PKI can check with the CA to update its list of revoked digital certificates so that it can fail the authentication of any revoked identities. This process can be manual or automated, depending on how the PKI is able to respond to each CAs revocation process. At a minimum, manually removing a revoked identity from a server key store or a revoked root CA certificate from a client certificate store is sufficient to handle the revocation once it is known.

5

Secure Sockets Layer (SSL)

The *Secure Sockets Layer (SSL)* provides data privacy and reliability over network connections using elements of Public-Key Infrastructure (PKI). It does so by allowing a client to authenticate a server in order to establish a connection between them and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data.

This chapter provides an overview of SSL including its standards, features, and requirements, and describes the elements of PKI that it employs. It also references how OpenEdge supports SSL and defines some SSL terms used in the OpenEdge documentation set.

For details, see the following topics:

- SSL standards support in OpenEdge
- Features and services
- SSL session components
- Support for trust
- SSL interactions and the user

SSL standards support in OpenEdge

OpenEdge supports SSL based on the following standards:

- *The SSL Protocol Version 2.0/3.0*, Transport Layer Security Working Group, Netscape Communications Corporation
- *The TLS Protocol Version 1.0*, Network Working Group RFC2246

- *Public-Key Infrastructure X.509*, PKIX Working Group

For an overview of Public-Key Infrastructure, see Public-Key Infrastructure (PKI) on page 69.

Features and services

SSL provides the following features and services:

- **Transport independence** — An application level wire-protocol that runs on top of a reliable transport protocol, such as TCP/IP.
- **Application independence** — Runs in any application environment as supported by vendor ports to different platforms. OpenEdge, in particular, supports SSL running under:
  - Internet application environments using HTTPS.
  - Several OpenEdge application environments, including the OpenEdge RDBMS, the AppServer, the WebSpeed Transaction Server, and the OpenEdge Adapter for SonicMQ BrokerConnect. For more information, seeSSL in OpenEdge on page 83.
- **PKI support** — Provides the ability for an SSL client to validate an SSL servers identity, which the server asserts in the form of a public key, so that the client can be assured of who it is communicating with. The server and client use the same key information used to assert the servers identity to securely exchange session-specific symmetric data encryption keys used to provide data privacy for the SSL session.
- **Limitations on PKI support** — The OpenEdge implementation imposes the following SSL limitations:
  - No support for optional PKI client authentication by the server
  - No support for checking the CAs CRL list to detect revoked SSL server digital certificates

SSL can have a significant performance impact on any enterprise network application. The OpenEdge implementation of SSL uses SSL session caching when possible to reduce the performance overhead that SSL connections incur (see Session caching on page 80. However, note that SSL can impose burdens on network application performance in any case.

Caution: Because of its performance impact, be certain that you need the security that SSL provides before you choose to design and build SSL into your application.

SSL session components

For SSL, the basic unit of secure network activity is the SSL session. An *SSL session* represents the security contract (key and algorithm agreement information) that occurs over a connection between a client (*SSL client*) that is connected to a server (*SSL server*) using SSL. An SSL session is generally governed by *security policies* that control the SSL session parameter negotiations between a client and server during the SSL connection process.

SSL servers and clients are generally configured at startup to follow certain security policies. For an SSL server, the most important policy is generally the SSL server identity that it assumes and that has been authorized by a trusted Certification Authority (CA). When an SSL client attempts to connect to an SSL server, the client authenticates the server identity. The session can begin after the client has successfully authenticated the server identity and the client and server have agreed on a set of security algorithms.

The SSL session continues until the client or server terminates it or the underlying TCP connection is broken. Until then, the session proceeds with the interaction among various SSL session components used by the SSL client and server.

Note: The SSL technologies are typically invisible to users of an SSL session.

SSL session components can be summarized as follows:

- Session identity on page 79
- Keys and certificates on page 79
- Algorithms (ciphers) on page 79
- Session caching on page 80

Session identity

A given SSL session, with all of its components, has a unique session ID that can be used to identify and maintain the session as a unit, independent of any current client/server connection.

Keys and certificates

The cryptographic keys and digital certificates that are used during an SSL session are:

- **Asymmetric keys** — The private (server) and public (client) keys used to allow an SSL client to authenticate the identity of an SSL server.
- **Digital (public-key) certificates and store** — Electronic documents used to store and access public keys and information for SSL client and server authentication. For more information, see Digital certificates and certificate store on page 73.
- **"Master secret"** — A secret session symmetric key shared between the SSL client and server that is exchanged using asymmetric cryptography and is used to generate the session-specific data encryption keys.
- **Symmetric keys** — Generated by the "master secret" and used for symmetric encryption and decryption of data exchanged between the client and server. These keys are unique for the current connection between an SSL client and server.
- **"MAC secrets"** — Generated by the "master secret," these are secret symmetric keys used for Message Authentication Code (MAC) operations on the client and server. The MAC provides the data integrity between the SSL client and server. These keys are unique for the current connection between an SSL client and server.
- **Other values** — Miscellaneous generated values to aid session cryptography and message transport.

Algorithms (ciphers)

Among the algorithms specified for an SSL session include those for:

- **Symmetric cryptography** — For encryption and decryption of bulk data sent between SSL client and server
- **Message Authentication Code (MAC)** — For message digest operations used to provide data integrity on either the client or server

Note: Some implementations also support data compression. However, OpenEdge does not provide data compression as part of its SSL support. The algorithm for asymmetric cryptography is defined when the private/public-key pair is initially created for a new SSL server identity.

Session caching

By setting an option on the SSL server, it can cache the SSL session (identified by its session ID) for a specified period of time after a given connection between the SSL client and server has terminated. During this specified period of time, if the same SSL client attempts to reconnect to this SSL server and requests a resumption of a previous SSL session (by ID), the usual authentication and hand-shaking required to establish an SSL session is dispensed with, and the SSL server reactivates the SSL session that is cached from the previous connection with this same client.

Note: The SSL server must be configured to allow session reuse. It might not be. The default OpenEdge configuration is to allow the SSL server to grant session reuse within three minutes. The default configuration for many SSL servers has session reuse turned off.

All the session components that were used in the previous connection are reused for this new connection except for those that are unique to a given connection (see Keys and certificates on page 79). This helps to minimize negative SSL performance impacts.

Support for trust

By implication, SSL supports a trust model required to establish trusted SSL server identities that is consistent with the PKI X.509 standard, and it advises due caution in the choice of root CAs for this purpose. In particular, SSL relies on the X.509 public-key certificates as authorized by root CAs and relies on the Digital Signature Standard (DSS) to assure SSL client and server authentication.

SSL also strongly suggests that any SSL implementation support certificate revocation messages and means for choosing a trusted root CA to authorize digital certificates, but does not directly specify how to do so. It also suggests that means be provided to view information about digital certificates and root CAs.

Note: OpenEdge does provide key and certificate management tools. For more information, see the sections on managing OpenEdge key and certificate stores in *OpenEdge Getting Started: Installation and Configuration*.

As stated in *The SSL Protocol Version 3.0* specification (see SSL standards support in OpenEdge on page 77), the "F.3 Final notes" section: "The system is only as strong as the weakest key exchange and authentication algorithm supported, and only trustworthy cryptographic functions should be used. Short public keys, 40-bit bulk encryption keys, and anonymous servers should be used with great caution. Implementations and users must be careful when deciding which certificates and certificate authorities are acceptable; a dishonest certificate authority can do tremendous damage."

SSL interactions and the user

Most user interactions with SSL happen during configuration of SSL clients and servers. This includes the maintenance of SSL server identities and client root certificates, as well as the setting of any SSL session properties prior to running a given SSL client or server.

During an SSL session, users (especially on the client side) might not be aware that an SSL session is in progress, though in practice applications that use SSL generally provide messages that notify users of their security situation. A common example is in Web browser applications that advise users of Web sites that they might visit for which proper SSL authorization cannot be established. The Web browser usually offers the option to forego accessing any unauthorized Web sites.

In OpenEdge, if an SSL client fails to authenticate an SSL server, the connection fails and a message is returned for handling by the client, which at a minimum displays a message for the user. For more information on OpenEdge support for SSL, see SSL in OpenEdge on page 83.

6

SSL in OpenEdge

OpenEdge supports SSL client and server connections over the Internet (using HTTPS and appropriate middleware) or on an intranet (using an OpenEdge implementation of SSL). This chapter describes how OpenEdge client and server components support SSL.

Caution: Use of SSL in any network environment can have significant performance impact. Be certain that you need the security that SSL provides before you choose to design and build SSL into any of your application networking.

For details, see the following topics:

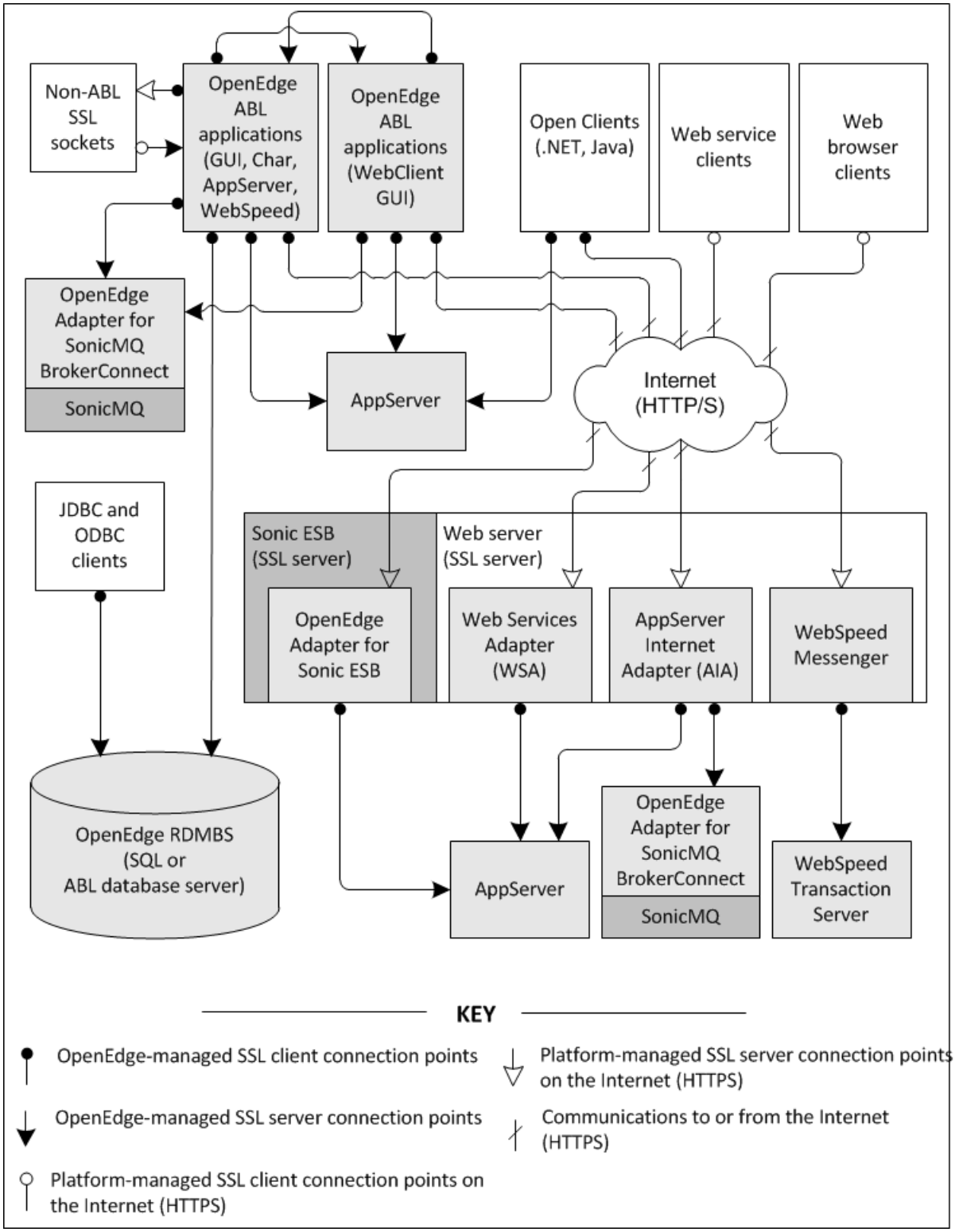
- OpenEdge SSL architecture
- Using SSL in OpenEdge
- Managing SSL server identity
- Configuring and running SSL sessions

OpenEdge SSL architecture

The following figure shows the general support for SSL in OpenEdge.

In the figure, SSL clients start from the top and access SSL servers moving toward the bottom. From this figure, you can see where OpenEdge supports authenticated and encrypted connections from an initial SSL client connection through intervening middleware and servers all the way to data access using the OpenEdge RDBMS. All of the indicated connection pathways that join components to each other or to the Internet support SSL in one form or another.

Figure 4: SSL architecture in OpenEdge



OpenEdge SSL client and server components

OpenEdge-managed SSL client connection points indicate SSL clients specifically supported by OpenEdge. These clients can be accessing OpenEdge servers using SSL, either indirectly on the Internet (using HTTPS) or directly on an intranet, depending on the server involved in the connection. OpenEdge-managed SSL server connection points indicate SSL servers specifically supported by OpenEdge for direct access on an intranet without the need for an intervening Internet connection.

All OpenEdge-managed SSL servers rely on a common OpenEdge key store to manage the private keys and server digital certificates required to support SSL connections from clients. These OpenEdge-managed SSL servers include the OpenEdge RDBMS, the AppServer, the WebSpeed Transaction Server, the OpenEdge Adapter for SonicMQ BrokerConnect, and ABL socket servers.

Similarly, most OpenEdge-managed SSL clients and servers rely on a common OpenEdge certificate store to manage the root CA digital certificates that enable them to establish connections to appropriate SSL servers. These OpenEdge-managed SSL clients and servers in ABL include database clients, socket clients, AppServer clients (including Web service clients), and SonicMQ BrokerConnect clients. Other OpenEdge-managed SSL clients include the SQL clients (JDBC and ODBC), AppServer Internet Adapter (AIA), Web Services Adapter (WSA), Sonic ESB Adapter, and WebSpeed Messenger. Exceptions include the .NET and Java Open Clients, which rely on their own certificate store facilities as supported by the Open Client Toolkit.

For more information on OpenEdge support for managing key and certificate stores, see Managing SSL server identity on page 86.

Note: For OpenEdge server components that have the option of using a NameServer to manage distributed server resources, the NameServer itself is never a party to an SSL connection. This is because the NameServer does not participate in the transmission of application data over any of the SSL client/server connections that it facilitates.

The OpenEdge SSL client and server connections shown within (or between) ABL applications indicate ABL applications acting as SSL clients, SSL servers, or both using ABL socket connections. Using sockets, ABL applications running on different ABL client machines can communicate securely with each other using the same SSL infrastructure as any OpenEdge SSL client and server (such as an ABL client and AppServer). Note, also, that ABL sessions can be socket servers for non-ABL socket clients and can be socket clients for non-ABL socket servers.

Note: An ABL session running in an AppServer or WebSpeed agent can function as an SSL socket client only (not as a socket server).

Non-OpenEdge SSL client and server components

Platform-managed SSL client connection points indicate SSL clients that rely on platform-specific SSL certificate stores to access the Internet using HTTPS. These are SSL clients that rely on no OpenEdge-specific management or features to access OpenEdge servers over the Internet. (Of course, where an OpenEdge client accesses an OpenEdge server over the Internet, it also relies on OpenEdge-specific features to do this.)

Similarly, platform-managed SSL server connection points indicate OpenEdge middleware that relies on platform-specific SSL key stores to manage SSL client requests from the Internet and pass them on to OpenEdge servers for processing. This OpenEdge middleware includes the WebSpeed Messenger and AppServer adapters that run on industry-standard Web servers (or Java Servlet Engines, JSEs) or, in the case of the Sonic ESB Adapter, that runs on the Sonic Enterprise Service Bus (ESB). Web server platforms and the Sonic ESB provide their own key store management.

The lightly shaded components in *Figure 4* identify the client, middleware, and server components provided entirely by OpenEdge. Unshaded components, such as the .NET and Java Open Clients or the JDBC and ODBC SQL clients, rely on some OpenEdge-provided interface code to allow the otherwise non-OpenEdge clients to access OpenEdge servers. Other unshaded components, including the non-ABL SSL socket (client or server), Web service client, Web browser client, and Web server components, represent third-party components that are configured with no necessary functional elements from OpenEdge.

The darkly shaded components represent other non-OpenEdge-specific technologies for integrating OpenEdge applications with non-OpenEdge applications. The SSL support provided by OpenEdge has no connection (other than the standard SSL support) to any security features that might be provided by these integration technologies. For example, Web service client access to OpenEdge services provided by the Sonic ESB Adapter is entirely managed by the Sonic ESB, as is access by other ESB services and clients. Likewise, the SonicMQ BrokerConnect supports direct and Internet SSL access from ABL, but the security of Java Message Service (JMS) transactions between the SonicMQ BrokerConnect and SonicMQ is entirely managed by the SonicMQ JMS infrastructure.

The rest of this chapter describes how you can work with the components described in this architecture, and where you can go for more information. For more information on SSL and its basic components, see Public-Key Infrastructure (PKI) on page 69.

Using SSL in OpenEdge

Using SSL in OpenEdge requires some preparation, depending on the connected components and the type of connection you are using.

To use SSL in OpenEdge, either over the Internet or for an intranet:

1. Establish the required SSL server identity and the means to authenticate it on all SSL clients that access the SSL server.
2. Configure or program, as appropriate, each SSL server and client component to connect using SSL over the Internet or on an intranet.
3. Start up each SSL server component before starting any of its SSL clients.

The remaining sections in this chapter describe key requirements and OpenEdge support for completing these steps.

Managing SSL server identity

Before you can enable an SSL connection of any kind, you must ensure that SSL servers and clients have access to the required keys and digital certificates to properly authenticate a connection and exchange encrypted communications over it. Each SSL server asserts its identity using a private key and server public-key certificate (*key store entry*) accessed from the servers key store. Each SSL client must successfully validate the servers identity using a corresponding root public-key certificate (*root certificate store entry*) accessed from the clients certificate store. The client and server also use their corresponding SSL key and certificate store entries to initiate encrypted communications between them.

OpenEdge provides a common key store for all OpenEdge-managed SSL servers and a common certificate store for all OpenEdge-managed SSL clients. This allows you to use a common set of SSL management tools to support your OpenEdge SSL infrastructure.

Note: OpenEdge provides separate certificate store management for Java Open Clients and relies on Microsofts certificate store management for .NET Open Clients. For Web service clients and Web servers (or JSEs), you must use the key and certificate store tools provided for the specific client or server platform. For more information, see *OpenEdge Development: Open Client Introduction and Programming* and the documentation for your SSL client or server platform.

You use these OpenEdge key and certificate stores to support both Internet and intranet SSL communications between SSL servers and SSL clients.

For more information on the tools for managing the common OpenEdge key and certificate stores, see *OpenEdge Getting Started: Installation and Configuration*.

Establishing SSL server identity

OpenEdge supports two basic options to establish an SSL server identity:

- *Using the default SSL server identity* — A common built-in SSL server identity installed with OpenEdge
- *Managing your own SSL server identity* — A unique server identity authenticated by a public or private Certification Authority (CA)

For each key store entry on an SSL server, you provide a unique, password-protected alias name in the OpenEdge key store; and for each corresponding root certificate store entry on an SSL client, the SSL management software generates a unique alias name (not password protected) in the OpenEdge certificate store.

A given server identity has a specified lifetime when it is valid. You therefore must update the key store entry for that identity with a new server public-key certificate that is authenticated by a trusted CA when the current certificate expires. At this time, you must also verify that the SSL clients root certificate store still contains a valid root digital certificate for the newly issued SSL server digital certificate and update it if necessary.

The sections that follow describe these options.

Using the default SSL server identity

The default key and certificates installed with OpenEdge support an SSL server identity that is common to all OpenEdge installations. These are appropriate for use on an intranet where SSL server authentication is not required, but where you still want to maintain data privacy over protected connections with a minimum of management. In other words, the default server identity is an easy-to-use SSL solution. Its primary purpose is in demonstrations and development environments, but you can also use it in intranet situations. For intranet production purposes and for use on the Internet, Progress Software Corporation recommends that you use a public or private CA-issued server digital certificate.

To use the default server identity with any OpenEdge server, you need only configure it to use SSL. OpenEdge automatically uses the key store alias, `"default_server"`, and the default password, `"password"`, unless you specify otherwise.

You can also update the `default_server` entry with a new trusted CA server certificate, adding effective server authentication to your default intranet SSL environment. If you use the same default password, you do not need to change any default server configurations for this update.

Note: The `default_server` identity installed with OpenEdge expires in the year 2040 to minimize any concern with its valid lifetime.

Managing your own SSL server identity

To manage your own OpenEdge SSL server identity and make it available to SSL clients, you must generate a password-protected, alias-named private key for the server and obtain a server certificate that is authorized by a trusted CA. The trusted CA can be one of the major public CAs, including RSA, Thawte, or Verisign; or it can be any other CA that you trust for your purpose, including your own internal CA. Follow the requirements of your chosen CA in order to request and receive the server certificate that you need. Once you have the authorized server certificate, you must install it, together with the corresponding private key, as an entry in the key store of any OpenEdge server you want configured with this SSL identity.

You then must propagate to all SSL clients (if necessary) the root CA public-key certificate that corresponds to the authorized server identity. OpenEdge comes installed with the root CA certificates for the major public CAs, including RSA, Thawte, and Verisign, that you can use to authenticate servers that they authorize. If you use another CA (including your own internal CA), you must appropriately obtain (or generate) the root public-key certificate and install it in each SSL clients certificate store.

Using the OpenEdge SSL management software, you can add, list, update, and debug SSL server identities defined in both OpenEdge-managed SSL server key stores and SSL client certificate stores.

Next, you need to configure standard SSL connection parameters and properties associated with each OpenEdge client and server component using a given SSL server identity in order to initiate and maintain SSL connections between them. For any SSL server identity other than the default, you must specify the key store entry alias name and password to configure the specified identity for a given SSL server. If you need to configure a server component manually (required for starting up the OpenEdge RDBMS), you must provide an encrypted form of the key store entry password during server configuration or startup.

Note: You can provide effective server authentication using the default server identity by updating the `default_server` key store entry with a new trusted CA server certificate. If you do not change the default password for this update, you can continue to use any default SSL configurations without change. However, if you change the password, you must then specify the new password for each SSL server configured using the `default_server` key store entry.

For more information on using the OpenEdge tools for managing SSL server identities and obtaining encrypted forms of key store entry passwords, see the sections on managing OpenEdge key and certificate stores in *OpenEdge Getting Started: Installation and Configuration*.

The remainder of this chapter describes how to configure OpenEdge SSL clients and SSL servers for a given SSL server identity.

Configuring and running SSL sessions

When an SSL server and client have access to authenticating keys and certificates they can engage in SSL communications, which occur in the context of an SSL session. Before starting them, you have to configure, code, or otherwise specify to the client and server that they must communicate using SSL as directed by certain SSL session properties. The mechanisms for doing this differ depending on the server and client type, but the functional and manageable properties of an SSL session are the same among all OpenEdge server types and among most OpenEdge client types. During an SSL session, clients that you program (ABL and Open Clients) can also access the identity of the SSL server they are talking with.

The following sections describe these SSL session properties:

- Configuring SSL server sessions on page 89
- Configuring SSL client sessions on page 91
- Accessing an SSL servers X.500 Subject Name on page 94

Each SSL session property has a functional name that differs slightly, depending on the component you are configuring and how you configure it. The possible mechanisms for specifying these properties include the:

- Startup parameters (or switches) on the command line or in scripts
- Unified Broker framework component property editors or component properties in the `ubroker.properties` file
- Connection parameters, object properties, and other appropriate internal code for SSL client or server components that you program yourself

The following sections describe the function of these SSL session properties using generic names and phrases and specify the mechanism that you can use to set them for each SSL server and client component.

Note: The actual property settings in OpenEdge might be expressed differently depending on the actual property and the mechanism used to set it.

Configuring SSL server sessions

The following table describes the SSL session properties that you can set for an OpenEdge SSL server.

Table 2: OpenEdge server SSL session properties

Property	Default setting	Description
SSL connections enabled	No	Specifies if all connections to this server must use SSL.
Key alias name	`"default_server"`	Sets the alias name of the key store entry used by the SSL server to assert its identify to the SSL client attempting to connect to that server.
Key alias password	`20333c34252a2137` (which is `"password"`, encrypted)	Sets the password to use for accessing the authenticating key store entry. You must specify a password when you specify the key alias name. The password must be encrypted unless you enter it in OpenEdge Management or OpenEdge Explorer for the given server component, where OpenEdge encrypts the password automatically. Otherwise, you must specify the password as an encrypted value that you can obtain using the `genpassword` command-line tool located in the `bin` directory of your OpenEdge installation. For more information on this tool, see *OpenEdge Getting Started: Installation and Configuration*.If you use the default key store entry, it also has a default password that you do not need to specify, unless you have changed it in a post installation update of the default server identity (see Managing your own SSL server identity on page 88).

Property	Default setting	Description
Use a session cache	Yes	Specifies if the server provides a cache for SSL client sessions that might be resumed after the client disconnects from the SSL server.
Session time-out	180 seconds	Specifies, in seconds, the length of time that an SSL client session is held in the session cache, during which an SSL client can resume its session. If the client has not reused or resumed a session within the specified amount of time, the SSL session information is discarded and the SSL client must make a full SSL session connection in order to resume access to his server.

The following table describes the mechanisms for setting SSL session properties for OpenEdge SSL servers and indicates where you can find more information about them.

Table 3: Mechanisms for setting OpenEdge SSL server session properites

This SSL server component . . .	Relies on this mechanism . . .	To set its SSL properties as described in . . .
OpenEdge RDBMS	Startup parameters	• *OpenEdge Data Management: Database Administration* • *OpenEdge Deployment: Startup Command and Parameter Reference*
AppServer	The Unified Broker framework	• *OpenEdge Application Server: Administration* • OpenEdge Management or OpenEdge Explorer online help
SonicMQ BrokerConnect	The Unified Broker framework	• *OpenEdge Application Server: Administration* • OpenEdge Management or OpenEdge Explorer online help

This SSL server component . . .	Relies on this mechanism . . .	To set its SSL properties as described in . . .
WebSpeed Transaction Server	The Unified Broker framework	• *OpenEdge Application Server: Administration* • OpenEdge Management or OpenEdge Explorer online help
ABL socket server	Connection parameters	• *OpenEdge Development: Programming Interfaces* • *OpenEdge Development: ABL Reference*

Other server platforms potentially involved with OpenEdge SSL clients, such as Web servers (or JSEs) and the Sonic ESB, provide their own means for configuring them to use SSL. For more information, see the platform-specific documentation.

Configuring SSL client sessions

The following table describes the SSL session properties that you can set for an OpenEdge SSL client.

Table 4: OpenEdge client SSL session properties

Property	Default setting	Description
Use SSL to connect	No	Specifies if the client connection uses SSL tunneling.
Reuse a disconnected SSL session	Yes	Specifies if the connection reuses a previously established and disconnected SSL session when the client connects to an SSL server.
Verify the SSL server host name	Yes	Indicates if the client compares the host name of the SSL server with the Common Name specified in the server digital certificate, and raises an error if they do not match. If this property is set to No, the client never raises the error. This can be a useful setting if SSL server identity is likely to be moved from one physical machine to another and you do not want this to interfere with making the connection.
Certificate store location (Java Open Client only)[1]	None	Specifies the location of the root certificate store used by the client for SSL connections.

[1] The certificate store for all OpenEdge-managed SSL clients, except the Java and .NET Open clients, is installed in the same location under the OpenEdge installation path for each OpenEdge SSL client. For Java Open Clients, there is no standard location for the certificate store, so you must manage it using the tools provided with the Open Client Toolkit and specify its location to configure Java Open Client SSL connections. For .NET Open Clients, you must locate and manage the certificate store as provided by the tools available with Microsoft .NET.

The following table describes the mechanisms for setting SSL session properties for OpenEdge SSL clients and indicates where you can find more information about them.

Table 5: Mechanisms for setting OpenEdge SSL client session properties

This SSL client component:	Relies on this mechanism:	To set its SSL properties as described in:
ABL database client	Startup and connection parameters	• *OpenEdge Deployment: Startup Command and Parameter Reference* • *OpenEdge Development: Programming Interfaces* • *OpenEdge Development: ABL Reference*
ABL AppServer client	Connection parameters (for the Internet using the AIA or an intranet)	• *OpenEdge Application Server: Developing AppServer Applications* • *OpenEdge Development: ABL Reference*
ABL client of industry Web services[2]	Connection parameters	• *OpenEdge Development: Web Services* • *OpenEdge Development: ABL Reference*
ABL client of the SonicMQ BrokerConnect[2]	Connection parameters (for the Internet using the AIA or an intranet)	• *OpenEdge Development: Messaging and ESB* • *OpenEdge Application Server: Developing AppServer Applications* (for URL formats and usage)
ABL socket client[2]	Connection parameters	• *OpenEdge Development: Programming Interfaces* • *OpenEdge Development: ABL Reference*

[2] ABL code in WebClient applications can also perform any of these SSL client functions in a user-downloadable application environment. However, note that the WebClient does not support access to any OpenEdge relational data source, including the OpenEdge RDBMS. The WebClient relies on the AppServer for all access to OpenEdge relational data sources. OpenEdge provides separate support for securely downloading and updating the WebClient and its application code over the Internet or an intranet. For more information, see *OpenEdge Deployment: WebClient Applications*.

This SSL client component:	Relies on this mechanism:	To set its SSL properties as described in:
.NET Open Clients of the AppServer	Connection parameters and run-time properties (for the Internet using the AIA or an intranet)	• *OpenEdge Development: Open Client Introduction and Programming* • *OpenEdge Development: .NET Open Clients* • *OpenEdge Application Server: Developing AppServer Applications* (for URL formats and usage)
Java Open Clients of the AppServer	Connection parameters and run-time properties (for the Internet using the AIA or an intranet)	• *OpenEdge Development: Open Client Introduction and Programming* • *OpenEdge Development: Java Open Clients* • *OpenEdge Application Server: Developing AppServer Applications* (for URL formats and usage)
JDBC or ODBC SQL database clients	Switches on the client connection string	• *OpenEdge Data Management: SQL Development* (sections on SQL-ABL interoperability)
AppServer Internet Adapter (AIA)	The Unified Broker framework	• *OpenEdge Application Server: Administration* • OpenEdge Management or OpenEdge Explorer online help
Web Services Adapter (WSA)	The OpenEdge Management or OpenEdge Explorer properties editor for each Web service instance	• *OpenEdge Application Server: Administration* • OpenEdge Management or OpenEdge Explorer online help
Sonic ESB Adapter	Sonic ESB tools for managing OpenEdge services	• *OpenEdge Application Server: Administration* • Sonic ESB documentation
WebSpeed Messenger	The Unified Broker framework	• *OpenEdge Application Server: Administration* • OpenEdge Management or OpenEdge Explorer online help

Other client platforms potentially involved with OpenEdge SSL servers, including Web service clients of OpenEdge Web services (managed by the WSA) and Web browser clients of WebSpeed, provide their own mechanisms for making SSL connections using HTTPS. For more information, see the relevant platform documentation.

Accessing an SSL servers X.500 Subject Name

Most user-programmable OpenEdge SSL clients, including ABL clients, .NET Open Clients, and Java Open Clients, provide a mechanism for you to access the authenticated SSL server name of most SSL servers directly involved in an SSL connection. The SSL server name is the X.500 Subject name in the SSL server certificate used to authenticate the SSL server to the client. This name is distinct from and often different than the network host name of the computer on which the SSL server runs.

For Internet connections, where the immediate SSL server is the Web server and not necessarily the final server endpoint for the client, the SSL server name returned is that of the Web server. This is true even if the Internet client ultimately accesses an AppServer that is accessed as an SSL server by middleware provided, for example, by the WSA or AIA.

This feature is a useful tool when you must disable SSL server host name verification (see *Table 3*), yet you want the application to verify the connected SSL server. Thus, when you disable host name verification, after each SSL connection to a server, you can use this feature to access the SSL server name and verify it manually against an internal list of acceptable SSL server connections.

The following table describes the mechanisms provided by supported, programmable, OpenEdge SSL clients for accessing the SSL server name and indicates where you can find more information about them.

Table 6: Mechanisms for accessing the SSL server X.500 Subject name

This SSL client component:	Uses this mechanism:	To access the SSL server name as described in:
ABL database client	`SSL-SERVER-NAME` function	• *OpenEdge Development: ABL Reference*
ABL client of an AppServer, Web service, or socket server	`SSL-SERVER-NAME` attribute on the AppServer, Web service, or server socket handle	• For AppServer clients, *OpenEdge Application Server: Developing AppServer Applications* • For socket clients, *OpenEdge Development: Programming Interfaces* • For all ABL supported SSL clients, *OpenEdge Development: ABL Reference*
.NET Open Clients	`_GetSSLSubjectName()` common Open Client object method	• *OpenEdge Development: .NET Open Clients*
Java Open Clients	`_getSSLSubjectName()` common Open Client object method	• *OpenEdge Development: Java Open Clients*

III

Auditing

For details, see the following topics:

- Auditing in OpenEdge
- Audit Security
- Configuring OpenEdge Auditing
- Developing an Audit-enabled OpenEdge Application
- Deploying an Audit-enabled OpenEdge Application
- Maintaining Audit Data
- Querying and Reporting on Audit Data

7

Auditing in OpenEdge

This chapter introduces the OpenEdge auditing core service.

For details, see the following topics:

- OpenEdge auditing
- Evaluating your auditing requirements
- OpenEdge auditing recommendations
- Auditing OpenEdge databases or clients earlier than Release 10.1
- Audit data storage management

OpenEdge auditing

OpenEdge auditing is a key core business service designed to help you:

- Establish secure auditing that supports nonrepudiation of audit data. You can see who did what, where, when, and how.
- Implement a separation of duty regarding audit data access by assigning privileges related to audit policy creation and management, audit data reading and reporting, and audit data archiving.
- Use a consistent solution for auditing across SQL, ABL, and the database utilities.
- Query and report on audit data records.
- Comply with international governmental regulations regarding the accuracy and reliability of company information.

- Archive audit data.

What you can audit

You can audit the following types of events:

- **Database events** — Record-level events (`CREATE`, `UPDATE`, `DELETE`)
- **Internal events** — Authentication (login), schema changes, audit policy administration, security administration, and database utilities
- **Application events** — Application context or application-defined events

Auditing database events

Creating, updating, or deleting records on a database table are all considered database events. If you want to audit a database, you must audit-enable the database, create a policy, configure table and field events, and enable the events whose audit data you want to record.

When you audit-enable a database, all internal tables used by OpenEdge auditing are created. However, if you audit-enable a database but do not configure a policy to audit any of its table or field events, no auditing takes place, even though the internal tables necessary for auditing are in place.

To audit database table and field events, you must set up auditing in the database where the schema for those tables and fields is defined; in other words, the audit policy for a databases events will always be recorded in the database where the tables and fields reside. The initial capture of the data must always be where the schema resides. You can later move the audit data into a designated archive, provided that you have audit data archiver privileges.

You can selectively configure OpenEdge database auditing per database, per database table, and per database table field. Within the database table and field categories, you can configure auditing for any combination of create, update, and delete operations, as well as control how much information is recorded about each of these operations.

For example, you can simply record the fact that something occurred in a table, or you can record what actually changed, as well as the old and new field values. By default, OpenEdge records field value information in a streaming format; you can optionally choose instead to record each single field value in a separate record.

When you set up database auditing, you can choose different audit-level settings for specific fields, even ones in the same table. Field-level settings always take precedence over table-level defaults. Any fields for which you **do not** specify particular audit settings retain the default settings for the table to which they belong; any fields for which you **do** specify individual audit settings are audited based on those unique settings.

If you configure auditing of database events, the audit data that is recorded provides the following information: which authenticated user was set for the database connection; the available database transaction IDs; the name of the table and field (if applicable) for which an operation occurred; what type of operation (create, update, or delete) occurred; and, optionally, the old and new values for any modified fields.

You can optionally include references to application context and user login context events captured by application auditing. This will give your database events context relating the database operation to an application operation. For more information, see Setting up OpenEdge auditing context on page 119.

Auditing internal system events

Tasks such as creating a user account, deleting a database administrator account, and updating a table trigger are all considered internal system events. When you audit-enable a database, the audit schema is created in the database and the internal event definition data is populated. What this means is that the name and the event ID for all OpenEdge-defined, internal system events are automatically loaded. (All event ID numbers below 32000 are reserved for OpenEdge events.)

From that point, it is up to you to audit-enable the events you want to record. In other words, although the Database record create event with the ID of 5100 is automatically added to the database event definition, you must specifically enable the audit event 5100 in the audit policy.

Provided with OpenEdge are several preconfigured audit policies that define field and table policy for internal events. These policies are similar to policies you would use to audit application database events. When you load the preconfigured policies, the policies are automatically activated. You can review the policies once you import them; then, if you want, you can deactivate any of them. You must then commit the changes brought about by importing and activating (and deactivating, if applicable) the policies in order to install those policies in each OpenEdge database and make them ready for use. Once you do so, auditing begins. For more information about these policies, see Preconfigured Audit Policies on page 235.

With the exception of audit archive events, all audited events are driven by table policy.

Auditing application events

Application events include application-context and application-defined events. You must enable application-context events and insert code into the application to trigger the events to be recorded. For application-defined events, you must define and enable the events, and then you must insert code into the application to trigger the events to be recorded.

You can define application events to be stored in the same database where an applications database events are recorded or in a database used specifically for application events; or you can record the events in any combination of an applications connected databases.

Once you enable application event auditing, you can audit the following:

- **Application-defined events** — Allows you to create your own application events for which there is no corresponding database operation; allows you to have full control over the granularity and detail of the audit data recorded; and allows you to group events into ranges for simplification of reporting.
- **Audit event groups** — Allows you to define (within the application) audit event groups, which can be used to establish a collection of related application or database audit events. For example, starting an update on a DataSet as part of a business task might change multiple table records in multiple databases. By starting and ending an audit event group around the DataSet `UPDATE` operation, you can later audit all audit event records that were involved in that single application operation.
- **Application context** — Allows you to record what application or business task the user was performing when the auditing event occurred. In viewing the context in which a change occurred, you can better determine whether the change was legitimate.

The application event and the application context audit data can be recorded together with the database audited events for consolidated reporting. The custom application audit data can also be managed fully by the supplied auditing tools for configuration, archiving, and reporting. For more information, see Configuring OpenEdge Auditing on page 117, Querying and Reporting on Audit Data on page 143, and *OpenEdge Data Management: Database Administration*.

Evaluating your auditing requirements

The OpenEdge auditing service simplifies the job of providing powerful auditing features to your application. The value you will get from auditing will be proportional to the amount of planning you apply to the configuration and implementation. There is a broad range of configuration options that support a number of application scenarios.

Application developer decisions

If you are an application developer working with auditing, you provide the application context audit code and make the architectural and storage decisions. You also make the following additional decisions related to auditing:

- Whether to include in your application-defined audit events those operations that do not result in physical database updates (for example, running a business task or pushing a button).
- How to divide or consolidate the audit policy that drives the run-time collection of audit data. You can choose a one-size-fits-all approach or a mix-and-match strategy.
- The storage locations of the application audit data.
- The application run-time (execution) context to apply to database record audits.
- The logical grouping of auditable application and database events that span multiple databases.
- How your application user authentication is designed to be able to record the correct application user ID in auditing.
- Whether to ship preconfigured audit policies as samples.
- Whether a customized audit policy maintenance tool should be made available to provide context to policy administrators who do not know the applications detailed schema.
- Whether to make available custom reporting tools that provide standard reports on common application data auditing.
- What tools, if any, to provide for automating audit data archiving and management.
- The particular auditing challenges you face, which are dependent upon:
  - The vertical market the application is running in. This affects the type of auditing to provide.
  - The individual production site security policies that drive auditing.
  - The specific network and hardware configurations at the production sites that control the long-term storage and handling of the audit data.
  - The perception of the amount of application context required to define database record events, which relates to how much forensic evidence is available for finding and fixing problems.
  - Regulatory compliance, regarding what data is captured and how long it is kept.

End user responsibilities

If you are an end-user working with auditing, you manage policies to meet your auditing needs and record what happened and where. You also decide:

- Which database table and field (records) to audit, provided you know the application database schema well enough.
- The amount of audit information to record to satisfy auditor requirements.
- The methods you want to use to manage the audit data's life cycle (onsite and offsite, online and offline).

OpenEdge auditing recommendations

Keep the following recommendations in mind as you prepare to implement auditing:

- Consider the application databases as short-term storage for audit data.
- Do not enable full audit indexes on short-term storage (apart from the databases used for reporting).
- Use a separate, Type II storage area for audit data and another separate, Type II area for audit indexes, and archive the audit data frequently.
- Use a designated database for long-term audit archiving and reporting, and make it more secure than the production database, which provides access to many users.
- Audit only what is absolutely necessary to save space and improve performance.
- Plan for reporting by grouping event IDs into ranges, structuring audit context field values consistently, and leveraging audit event groups.
- Tune your short-term auditing database for high performance in a high-traffic scenario.
- Tune your long-term storage for great size and reporting capabilities.

Auditing OpenEdge databases or clients earlier than Release 10.1

You can upgrade an OpenEdge Release 10.0 (or later) database to support auditing.

An OpenEdge or Progress client earlier than Release 10.1 will be unable to connect to an audit-enabled Release 10.1 OpenEdge database. The reason for this is that when you audit-enable a database, you add audit tables to the database. When the client communicates with the database, the database requires the client to be aware that auditing is enabled. In other words, the client must be audit-aware. Pre-Release 10.1 clients are unaware of auditing, so the connection is disabled.

If a database is audit-enabled, and later auditing is disabled, a pre-Release 10.1 client will then be able to connect to the database after the contents of the audit tables are entirely deleted.

Audit data storage management

Before you begin auditing, you should decide what kind of audit data storage solution you want to implement. Making this decision ahead of time, before you are faced with an accumulation of audit data in need of storage, eliminates having to find a suitable solution under pressure.

Within the context of auditing, **short-term** audit data storage refers to the auditing data tables of an OpenEdge database where the audit records are initially created. The short-term storage is configured to run very fast because it has no requirements to be the source for an audit report generator or to store large amounts of audit data.

Long-term audit data storage refers to the auditing data tables of an OpenEdge database that is dedicated to the storage and long-term management of audit data. Long-term storage is configured for optimal performance for storing large amounts of audit data (that has been consolidated from many sources), generating complex audit reports that require full indexing support, and providing additional security.

You can use the audit data tables in your applications database for long-term storage and keep the data permanently in the same database as your application data. For reasons related to providing security, maintaining audit data tables of a reasonable size (which affects performance), and making it easier to report on the data, it is recommended that you move the audit data recorded by each database to a separate database that is configured for that purpose. However, if the added security is not required and there are no size, performance, or reporting concerns, there is no functional reason why you have to move the data.

You can record your applications audit events with those recorded by the OpenEdge database. Direct the applications audit events to specific databases by loading the appropriate audit policy into each database, or direct it to the long-term storage.

For more information about audit data storage and archiving, see the relevant sections in *OpenEdge Data Management: Database Administration*.

8

Audit Security

When you audit data, you do so to obtain an accurate picture of who did what, when, where, and how. You might need this picture to satisfy internal or external requirements; in either case, security of the data collected is essential. Your auditing solution must provide you the means to control and manage the data you audit, preserve its integrity during storage, and prevent it from being tampered with or modified in any way.

In addition, the security provisions you implement must be applicable across all potential avenues of entry to the audit data: from SQL, an ABL client, and the database utilities. To accomplish this, OpenEdge employs an authorization solution based on the granting and revocation of auditing privileges, which allows you to manage audit access and separate audit duties.

For details, see the following topics:

- Overview
- Managing audit privileges
- Audit data archival security
- Audit security for database clients, tools, and utilities
- Audit security for OpenEdge databases
- Audit policy security
- Choosing additional audit options
- Configuring additional user authentication systems and domains
- Controlling the user ID and login sessions in audit data records
- Audit archiving
- Assigning a database a unique identifier

Overview

In data communications, *nonrepudiation* for a sender ensures that the entity that sends a message cannot later deny having sent it, and for a receiver ensures that the entity that has received a message cannot later deny having received it.

Nonrepudiation is also an essential component in audit data security. A user must not be able to deny having performed an action if it is recorded in the audit record; similarly, a user must not be able to claim to have performed an action if the audit record does not reflect that fact.

Asserting user identity

For auditing data integrity, the identity of the user who is recorded in the audit records must be absolutely reliable. That is, the user ID must be successfully authenticated by a trusted authentication system that OpenEdge recognizes.

OpenEdge supports the following sources of user accounts to authenticate a user ID for auditing:

- OpenEdge RDBMS `_User` table accounts available through the built-in `_oeusertable` authentication system
- Local operating system accounts through the built-in `_oslocal` authentication system
- External user accounts accessed by an ABL authentication callback defined for a user-defined authentication system
- External user accounts accessed directly by the ABL application and associated with a user-defined authentication system or the built-in `_extsso` authentication system

For more information, see *OpenEdge Getting Started: Identity Management* and *OpenEdge Development: Programming Interfaces*.

Adding security through separation of duty

Implementing an auditing solution requires making decisions about who will do what with regard to creating, updating, and deleting audit policies; reading audit data records; archiving audit data; and performing other auditing-related activities. You must consider whether you want all responsibility for auditing to reside with one individual, or whether you would rather establish a separation of duty.

The premise behind incorporating a separation of duty is that it removes total responsibility from only one individual and instead shares the responsibility for who can create, manage, maintain, or access auditing in some way. Administrative responsibility for audit data and policy can be separated from the administrative responsibility for the database.

OpenEdge provides four database-level auditing privileges. The auditing security model also allows control over whether privileged users can grant their own privileges to other users. Each user must be authenticated and authorized prior to being assigned auditing-related privileges.

The level of auditing privileges that a user possesses relates directly to the privileges the user has with regard to the audit policy and physical audit data tables. All the audit data that is collected, as well as the audit policy that controls what data is collected, is stored in internal tables. Based on the assigned privileges, these tables are accessible by the database utilities, ABL, and SQL when appropriate permissions have been granted.

The standard OpenEdge database `CAN-*` permissions do not apply to audit data and audit policy tables. The auditing privileges replace the `CAN-*` permissions.

Managing audit privileges

As you implement and maintain application-event, internal-event, and database auditing, you want to be sure that the audit data trail is complete and unalterable. To ensure that the audit data is complete, you create and activate audit policies, taking care to include as auditing events only those events whose occurrence is important to you. To ensure that the trail is unalterable, you determine who can access the audit data, and what form that access can take.

For example, who can create and activate an audit policy? Should that same user also be responsible for archiving and loading audit data, or even for determining when those tasks should be done? Do you want to make sure that someone cannot alter data and then cover that activity by altering the corresponding audit data record? Additionally, when should audit data be moved from short-term to long-term storage?

In other words, should the entire responsibility for creating, monitoring, and maintaining audit data rest with one individual? In all likelihood, no. For this reason, OpenEdge allows you to assign auditing roles to individual users and user accounts, thereby granting a user some (but not necessarily all) privileges with regard to audit policy configuration and audit data maintenance.

Inheriting audit privileges: the security administrator

When you initially install and start OpenEdge, it is the security administrator who, by default, can perform auditing operations. In general, audit privileges encompass policy administration, audit data administration, audit data access, and the ability to manually generate auditing events. Until such time as audit administrator privileges are assigned to anyone else, the security administrator can perform all audit policy functions.

Once the security administrator grants another user audit administrator privileges, however, the security administrator is no longer able to maintain the audit and policy data. The security administrator still retains control over the databases physical structure. For example, the security administrator can no longer audit-disable a database, create or update an audit policy, or archive audit data.

From that point forward, the administration of the audit data and policies is separate from the administration of the entire database. The security administrator is responsible for access control security for the general application data and has the ability to control user access and, for the database `_User` table accounts, the user accounts themselves.

In some cases, it might be that the audit administrator and the security administrator are the same person; however, in a case in which there is one audit administrator and one or more security administrators, it is only the audit administrator who can create, update, and delete audit policies. Audit administrators (and other audit privilege holders) can also grant their privileges to other users.

Assigning audit security privileges

By default, OpenEdge applies a GRANT authorization model to all audit-related database tables. This means that in order for an individual to be able to create audit policies and manage audit data, the individual must be granted the appropriate privileges to do so.

Because you might not want only one individual to have responsibility for all audit-related activities, you can assign to certain users one or more auditing privileges. When you assign privileges to a user, you also decide whether that user can then grant the same privileges to other users. Only users who have been granted the appropriate privileges can perform the corresponding auditing functions.

There are four audit security privileges:

- **Audit administrator** — An authenticated user who has been granted privileges to create, update, and delete audit policies and read audit data.

- **Application audit event inserter** — An authenticated user who has been granted privileges to generate application audit events. Note that in ABL applications, application of this privilege is optional and disabled by default; in SQL applications, application of the privilege is enabled by default and cannot be disabled.

  The application audit event inserter does not have privileges to archive audit data or policy tables.

- **Audit data archiver** — An authenticated user who has been granted privileges only to archive or load audit data. An audit data archiver has no access to audit policy.

- **Audit data reporter** — An authenticated user who has been granted privileges to read the audit data.

The audit administrator has unrestricted read access to all the audit tables; **no one** has the privilege to update the audit data, and only the audit data archiver can truncate or move the audit data to another location, maybe for long-term storage, for example. The audit administrator is the only user authorized to configure audit policy. The generated policy and audit data is stored in standard OpenEdge database tables, which allows you to easily query the data for audit details.

The addition or removal of a user account from the list of privileged audit users results in an auditing record being generated to preserve any and all changes.

As shown in the following table, a user who is granted a particular auditing privilege can (with permission) grant one or more audit privileges to other users. Whenever an audit administrator grants or revokes an audit privilege, that action is recognized system-wide by both the SQL and the ABL clients.

Table 7: Granting audit privileges to other users

A user with this audit privilege . . .	Can grant this privilege to other users . . .
Audit administrator	Audit administratorApplication audit event inserterAudit data reporterAudit data archiver
Application audit event inserter	Application audit event inserter
Audit data reporter	Audit data reporter
Audit data archiver	Audit data archiver

SQL administrators grant audit-related privileges through the SQL GRANT statement. ABL administrators use either Data Administration or the character Data Dictionary.

For more information, see the Database Administration online help, the Data Dictionary online help, *OpenEdge Development: Basic Database Tools*, and *OpenEdge Data Management: SQL Development*.

If no specific audit administrator is defined, the security administrator or ABL administrator automatically inherits the audit administrator privilege. If no specific security administrator or ABL administrator is defined, all users are, effectively, security administrators or ABL administrators and inherit the privilege of audit administrator.

Creating a primary and a secondary audit administrator

A recommended best practice is to create at least two audit administrators, one who is primary and another who is secondary. Keep in mind that once you name an audit administrator, the security administrator no longer has the ability to administer auditing or grant another user account audit administration or any other audit privileges. By naming two audit administrators, you avoid losing audit data access in the event of an audit administrators unforeseen unavailability or absence.

If there is a single audit administrator and access to that account is somehow lost, however, the security administrator can regain control of auditing by revoking the last audit administrators privileges and then retaking full control. Once control is restored, the security administrator can designate a new audit administrator.

Granting audit privileges

A security administrator grants the **initial** audit privileges; the administrator inherits this responsibility simply by being a security administrator or an ABL administrator. The specific steps the administrator follows differ depending on whether the administrator is working in a Windows or on a UNIX platform, and whether the administrator is an ABL security administrator or an SQL DBA.

At the time the first audit administrator is created, any audit privileges granted to user accounts by the security administrator will remain in effect until they have been specifically revoked.

This section provides an overview of the grant process. For the specific steps, see the following additional sources of information:

- If you are an ABL administrator in a GUI platform, see the Data Administration online help.
- If you are an ABL administrator on a UNIX platform, see either the Data Dictionary online help or *OpenEdge Development: Basic Database Tools*.
- If you are an SQL DBA, see *OpenEdge Data Management: SQL Development*.

Granting audit privileges in ABL

To grant audit-related privileges, you identify the user ID, select an audit privilege for the user, and add any optional comments. You also determine whether that user can grant privileges to other users. Any user with audit administrator privileges can grant all four privileges (including audit administrator) to other users.

Note that for all privileges except audit administrator, the **Can Grant Permissions for** option is, by default, not selected. The exception to this is when a user is being granted audit administrator privileges; an audit administrator automatically receives the right to grant the audit administrator privilege, or any of the other privileges, to other users. For this reason, the **Can Grant Permissions for** option is disabled when a user is being granted audit administrator permissions.

Remember that once the security administrator grants audit administrator privileges to a user, the security administrator no longer has audit administrator privileges. The exception to this is when there is only one audit administrator, in which case the security administrator can revoke that users audit administrator privileges (in the case of an emergency). This allows the security administrator access to the audit data if a situation arises in which the sole audit administrator is unexpectedly unavailable; without this provision, no one would be able to access the audit data.

Although the security administrator can revoke the users audit administrator privileges, the security administrator cannot revoke any of that users other permissions.

Granting audit privileges in SQL

To grant audit-related privileges in SQL, you use the SQL GRANT statement.

Once a SQL DBA grants the audit administrator privilege to a user:

- The SQL products DBA will retain the ability to SELECT (read) on all audit policy and databases.
- The SQL DBA will retain the ability to revoke an audit privilege for a user account that was granted by an audit administrator.
- SQL users will retain the ability to SELECT on the audit tables, if they have audit read privileges.

For details about the SQL GRANT statement, see *OpenEdge Data Management: SQL Development* and *OpenEdge Data Management: SQL Reference*.

Revoking audit privileges

Only the user who initially granted an audit privilege to another user can revoke that specific audit privilege. When a users privilege is revoked, all privileges granted by **that** user to other users are also revoked, unless an audit administrator is doing the revoking. In that case, the audit administrator can choose to revoke some user privileges and preserve others.

For example, consider that user A (who is not an audit administrator) has been granted the audit data reporter and the audit data archiver privileges, along with the ability to grant these same privileges to other users. User A grants the audit data reporter and the audit data archiver privileges to user B, who can also grant the privileges and does so to user C, who can do the same, and does, for user D.

If user As audit data reporter privilege is later revoked, the same will be true for users B, C, and D. However, if user As audit data archiver privilege is not revoked, users B, C, and D retain that privilege as well.

If user A had instead been an audit administrator, and that privilege was later revoked, all privileges of **every** type that user A had subsequently granted can (but not necessarily must) also be revoked in the same cascading fashion. When user A clicks **Revoke**, user A is prompted to confirm the action and then is able to choose which permissions to revoke for which user ID. User A can revoke user Bs audit data reporter privilege, for example, but allow user C to retain that same privilege.

Note: Changes to a user accounts auditing privileges take effect the next time the user establishes a client session with the OpenEdge database.

For the specific steps involved in revoking privileges, see the following additional sources of information:

- If you are an ABL administrator in a GUI platform, see the Data Administration online help.
- If you are an ABL administrator on a UNIX platform, see either the Data Dictionary online help or *OpenEdge Development: Basic Database Tools*.
- If you are a SQL administrator, see *OpenEdge Data Management: SQL Development*.

Audit data archival security

Audit data is initially recorded in the database in which the audit event occurred. This storage solution is often a short-term one, since the growth of the stored audit data can, over time, affect the overall performance and efficiency of the databases applications and the generation of audit reports.

A solution is to identify a designated database that will maintain the audit data for one or more databases over a longer term. To move audit data from short- to long-term storage, a user must have the audit data archiver privilege. The audit data archiver can then move the data by using either an archive application that the user has developed or the Audit Archiving utility supplied by OpenEdge.

The Audit Archiving utility uses data integrity checking to ensure that data is not lost or corrupted in the process. The utility also has the optional ability to validate the data integrity of each audit data record.

OpenEdge Audit Archiving

The Audit Archiving utility supplied by OpenEdge authenticates using any of the following methods:

- The user ID of the current operating system process must have the audit data archiver privilege. (No specific user credentials are supplied.)
- The `-userid` and `-password` parameters on the command line must match a user account in the `_User` table or operating system accounts.

- The `-userid` and `-password` parameters must still be authenticated even when no `_User` table accounts exist. The user ID must be an audit data archiver user, and the password must match the database MAC key.

Each time an audit archive is executed, the event is recorded in the audit data.

Audit security for database clients, tools, and utilities

In order for database clients to access the audit tables, the clients must be audit-aware. Being audit-aware means that the client understands audit schema and can distinguish an audit-related table from an application or built-in database table.

When a database client recognizes an audit table, the client uses separate access control tables designed specifically for auditing.

From the audit privileges, the client determines whether the user (the application) has the ability to perform `READ`, `CREATE`, or `DELETE` operations on the audit tables. `UPDATE` is never allowed on any audit data table; `UPDATE` is allowed only on those tables that contain audit configuration and policy information.

The OpenEdge database utilities protect the audit tables by:

- Allowing only the audit data archiver to copy, move, or delete audit data
- Recording each archive, copy, backup, recover, roll forward, dump, and load event into the audit tables
- Recording all changes in auditing configuration or administrator roles
- Using a MAC when dumping audit and policy tables to preserve data integrity
- Confirming the message digest of an audit data dump before loading it into a database

The database utilities also contribute to the data integrity of the audit information. When the database utility recognizes that the task it is to perform requires permissions to access audit tables, the utility will perform a series of actions to determine whether the user has the required audit permissions.

Audit security for OpenEdge databases

Each new database instance is created with a unique value (UUID) to clearly identify the database source of any audit table records. When a database is being moved or copied, the security administrator responsible can assign a new, unique identity to the new instance of the database. The generation of a new database identity preserves the point-of-access integrity in the audit data records. When each database instance has a unique identity, you can use the audit data to trace access to a particular instance and user.

Note that a read-only database copy can retain the original instances value, since it does not qualify as a unique database instance.

You can enable auditing on a Release 10.1 or later OpenEdge database. After auditing is enabled on an OpenEdge database, database connections require Release 10.1 or later ABL clients (including the AppServer and WebSpeed Transaction Server).

Audit policy security

Audit Policy Maintenance enables you to create and maintain audit policies (from simple to more complex) for application events, database events, and internal (to OpenEdge) system events. The exception to auditable events is any event that is hard coded and cannot be disabled through policy, such as when an audit archive or audit load, for example, is being done, since disabling these could compromise the security of the data.

In order to create and activate an audit policy, you must be an authenticated audit administrator, or a security administrator with audit administrator privileges. Granting of these privileges occurs within Data Administration or the Data Dictionary for ABL administrators and through the SQL GRANT statement for SQL administrators. For more information, see the Data Administration online help, *OpenEdge Development: Basic Database Tools*, or *OpenEdge Data Management: SQL Development*.

Sealing the audit data

You can optionally seal the audit data records by using either a message digest or a message authentication code (MAC). The message digest allows detection of unauthorized changes if someone attempts to modify audit data outside of an ABL or SQL application. The MAC is an encrypted value that is stored and displayed as a password field; it is also known as the DB Passkey. The MAC protects data integrity when data is being moved outside of the OpenEdge utilities.

Note: Only OpenEdge tools can verify the data seal.

When you create a policy, one of the settings you choose is the data security level, which controls the level of security applied to the audit data transaction tables `_client-session`, `_aud-audit-data`, and `_aud-audit-data-value`, each of which has a field called `_Data-seal`. The `_Data-seal` field holds an internally generated seal on the data that guarantees the integrity of the data and ensures it has not been tampered with outside of the registered services permitted to maintain this data.

The setting of the data security level determines what information is written to the `_Data-seal` field in each one of the audit data transaction tables, as follows:

- **No Additional Security** — No additional security is applied to audit data
- **Message Digest** — Stores a message digest in the `_Data-seal` field of the audit data tables to guarantee the integrity of the data
- **DB Passkey** — Stores a MAC in the `_Data-seal` field of the audit data tables to guarantee the integrity of the data (a MAC is more secure than a message digest)

Resolving audit policy conflicts

You can create and activate one or more policies at the same time, for the same data. Multiple policies merge to create one run-time policy.

Because the policies that are merging might specify different data security levels, different event settings for the same event, or different table and field settings for the same table/field combination, a conflict might develop between or among active policies. At any time, you can run a report (using Audit Policy Maintenance) that identifies conflicts between or among active policies. This feature is especially useful because you can see if any conflicts exist before you commit changes to a policy.

Within a single database, multiple policies can be active, with different data security levels. The data security level used in each case will be taken from the active audit policy that the event or table is defined in, depending on the type of event.

When a policy merge occurs, the maximum data security level that applies in each case is used.

The rules are as follows:

- **Avoid conflicts.**

  Promote a best practice of avoiding conflicts as much as possible. Define event policy under a separate policy name to tables and field policy, and load only a single event policy for standard events. Break out your application events into separate policies. Do not load multiple active policies with duplicate table or field settings; keep things separate.

- **Always resolve conflicts to the maximum security or detail level.**

  In conflict situations, it is much safer to use a model that results in a higher security or detail level than anticipated than for the merge to result in a potential security breach or missing data.

- **Conflicting identifying fields will be aggregated.**

  If multiple active policies have the same table policy with different identifying fields, the identifying fields will be aggregated and the resultant full list will be used. If conflicting ordinal positions occur, the order in which the identifying fields will be placed into the audit context is unpredictable.

- **If table policy event IDs conflict, the first instance of the event ID takes precedence.**

  If multiple active audit policies define the same table policy with different event IDs specified for the create, read, update and delete file operations, it is unpredictable which will get used. The tools should avoid this situation, if possible, and report on it when it occurs. No run-time error will result; however, it will simply use the first event ID it finds for a specific file.

- **The audit event takes precedence over any table or field settings.**

  For table and field level events (for example, create, delete, and update), the audit event policy determines whether auditing is on or off for the event. If auditing is off for the event, the table and field policy settings are ignored.

  Note that the level of detail on the event is irrelevant for table or field policy; it simply determines whether the event is on or off, as the detail level is defined for the table or field itself.

- **Within a single database, an event can have only a single setting.**

  An event is completely off, on with minimum details, or on with full details recorded. If the same event policy is found in multiple active audit policies in a database, the maximum level will be used for everything. This rule should always be resolved first to determine if auditing is on for the event or not, since the event takes precedence.

- **Within a single database, a table/field can have only a single setting.**

  If the same table or field is found in multiple active policies, then the maximum level will be used in each case. Note that when a table policy is defined, there is an implicit definition for all of the fields for that table with a level that matches the table level; any specific field definitions are exceptions.

- **Use the maximum data security level that applies in each case.**

  Within a single database, multiple policies can be active with different data security levels. The data security level used in each case will be taken from the audit policy that the event or table is defined in, depending on the type of event.

  For table events (for example, create, update and delete), the data security level will come from the policy the table belongs to. For nontable events (for example, authentication) the data security level will come from the policy the event belongs to. For the data security level, fields are irrelevant because there is an implicit field definition for every field in a table.

  Where a conflict occurs (for example, if the table or event is defined in more than one active policy), the maximum data security level from all the active policies the table or event is defined in will be used. For table events, a conflict situation develops only when the table is included in more than one policy; duplicate events are irrelevant, since the data security level is being used from the tables policy.

For more information, see the Audit Policy Maintenance Help.

Exporting audit policy configuration as an XML file

As an audit administrator, you can export an audit policy configuration as an XML file. Audit Policy Maintenance seals the XML file for security reasons, which prevents anyone from editing the file and then tricking someone into reloading it into the database as corrupted.

The only valid way of making an audit policy configuration change is through Audit Policy Maintenance. If a user is going to load the XML file (through either Audit Policy Maintenance or Data Administration), the tool checks the seal information before loading an XML file. If the seal information is invalid or missing, the tool does not load the XML file.

Choosing additional audit options

If you are working with an audit-enabled database, you can select two additional auditing options. You make these selections from the **Admin** > **Database Options** menu in Data Administration.

The options are as follows:

- **Use Application User Id for Auditing** — Overrides the database connections user ID for recording audit user ID and uses the application-level user ID for auditing (if set). If you select this option, you must also have set the user ID through the `SET-CLIENT( )` method. For details, see the Auditing chapter in *OpenEdge Development: Programming Interfaces*.
- **Enforce Audit Insert Privilege** — Tells the OpenEdge client executable to require that users of applications that use application auditing and set auditing context must be granted the audit inserter privilege in order for auditing data to be recorded.

Configuring additional user authentication systems and domains

Interwoven with data integrity in a successful auditing implementation is nonrepudiation. Nonrepudiation means that users cannot call into question their having performed an action if the action, in turn, generates a recorded audit event. Audit data that identifies what happened, where, and how is useful, yet incomplete if it fails to identify who performed the auditable application or database operation.

Each user ID specified in audit records must originate from a successful authentication to a user authentication system that OpenEdge trusts. It must be possible for the users authentication to be validated to ensure that the user ID was not compromised during transit from the authentication system to the database connection where the user ID is inserted into audit records. Only then can the user recorded in audit records be trusted to be accurate.

Prior to OpenEdge 11.0, the only trusted user ID source for OpenEdge applications was the `_User` table. However, reliance on the `_User` table meant that it was not possible for OpenEdge to validate that a user ID was coming from a trusted ABL application or other source. At issue was how an ABL application could use an external (to OpenEdge) authentication system and convey the resulting authenticated user ID to an ABL client in a manner that ensured that OpenEdge can trust the user ID in audit records.

The solution is to:

- Provide built-in authentication systems that allow OpenEdge to authenticate user identities against the `_User` table and local operating system accounts
- Support user-defined authentication systems configured with ABL authentication callbacks that allow OpenEdge to authenticate user identities against external user accounts
- Support both built-in and user-defined authentication systems that allow the ABL application to become a trusted source for user authentication.

With the availability of an ABL client-principal object (ABL security token) and a trusted domain registry, OpenEdge can thus establish a trust relationship with either `_User` table, local OS, external, or ABL application-managed user account systems, and OpenEdge can accept as authentic all user IDs from these sources.

User authentication based on the `_User` table (using the `-U` and `-P` parameters or the `SETUSERID` function) also remains valid and can be used as the user ID in the auditing records.

Note: For this version of OpenEdge, only the `_User` table or the local OS user account systems are available for authentication in SQL.

If you are connected as a security administrator, you can:

- Define your own authentication systems implemented either through your ABL application or through ABL authentication callbacks that you write and that OpenEdge can call directly to access external user account systems and otherwise manage the authentication process
- Customize the built-in authentication systems provided by OpenEdge using ABL authentication callbacks that you can write to modify the built-in authentication process
- Define domains that are configured to authenticate using a particular built-in or user-defined authentication system

If your ABL application implements its own authentication system, you must also make sure that:

- The domain specified for a client-principal object is included in the trusted domain registry.
- The application seals the client-principal object in the LOGIN state to begin a login session for a successful user authentication, or seals the object in the FAILED state for an unsuccessful user authentication.

For more information, see *OpenEdge Getting Started: Identity Management* and *OpenEdge Development: Programming Interfaces*.

Note: If you continue to use the OpenEdge `_User` table for authentication, you need not do anything further to set up authentication. Also, connecting to an OpenEdge database by using either no user ID (the default connection identity) or the User ID (`-U`) and Password (`-P`) connection parameters is audited automatically when you have the audit service enabled.

Controlling the user ID and login sessions in audit data records

As described in Configuring additional user authentication systems and domains on page 112, and in *OpenEdge Getting Started: Identity Management*, there is more than one way to architect an ABL application's security system to authenticate and authorize application/database access, which supports non-repudiation of the user ID recorded in auditing events. You might want to use a single user ID for the application and all database connections, or you might want to use different user IDs for the application and each database connection. Another option is to use a combination of the previous two. For this reason, an ABL application must be able to configure auditing to use the user ID set in the database connection or obtain it from an ABL session or application-level authentication operation.

Both the model of user authentication through `SETUSERID` and the model using the client-principal object are supported. The general rule, however, is that you employ only one of the models in an application.

When using the `SETUSERID` model, your application is required to connect to an OpenEdge database using either the `-U` and `-P` parameters or a default connection identity, and then call the `SETUSERID` function. Either way, your application is required to populate and use the user accounts provided by the OpenEdge database's `_User` table. With regard to auditing, the database connection's user ID is recorded in auditing records and the users domain is always the default (blank) domain.

The client-principal model gives the application much more flexibility, but might require some additional configuration work at the production site by the security administrator. Using this model requires at least some of the following steps:

1. Coding the ABL application to generate a client-principal object or retrieve an OpenEdge-generated client-principal object
2. Configuring the local domain registry in the OpenEdge database
3. Configuring the database options to set the authentication trust relationships allowing the database connection to trust a client-principal validated against one of the following domain registries:
   - Local database domain registry
   - Application (ABL session) domain registry
4. Coding the ABL application's initialization and use of a session domain registry. For example:
   - Loading the session domain registry from the local domain registry of an OpenEdge database
   - Building the session domain registry from an external, ABL-brokered domain source
5. Coding the ABL application for where a user identity must be set. For example:
   - Using the `SECURITY-POLICY:SET-CLIENT( )` method to validate and set an application-wide user identity, including for all available database connections
   - Using the `SET-DB-CLIENT` function to explicitly validate and set a user identity for one or more OpenEdge database connections only

Whether the application uses a single user identity (client-principal) for the entire application or multiple user identities is totally at the discretion of the application designer. Progress Software recommends that an application run with only one user identity at a time. The one rule is that the `SET-DB-CLIENT` (or `SETUSERID`) function supersedes any user identity set using the `SECURITY-POLICY:SET-CLIENT( )` method. For additional details about how to set this up in an application, see *OpenEdge Development: Programming Interfaces*.

The Audit Policy allows the audit administrator to select whether an authentication audit event will record just the simple user authentication operation or add the additional user login session information about the authentication system that was used.

The recorded user ID in the auditing records can come from either the ABL session for those applications that provide their own user authentication or from the OpenEdge database connection for those applications that use OpenEdge user authentication. You control where the user ID comes from through configuration options in the OpenEdge database that control that database's policy. When the database option **Use Application User Id for Auditing** is not selected (which is the default), the user ID recorded in auditing records can be set through one of the following three methods:

- The OpenEdge database `-U` connection parameter
- The ABL `SETUSERID` function
- The ABL `SET-DB-CLIENT` function, which requires a client-principal object

When the **Use Application User Id for Auditing** option is selected, the user ID recorded in audit records is taken from the ABL session, which is controlled through the `SECURITY-POLICY` object. On that object is a method `SET-CLIENT( )`, which allows you to set the session user ID using a client-principal object.

For more information about:

- Managing user identities, see *OpenEdge Getting Started: Identity Management*
- Creating and using a client-principal object, see *OpenEdge Development: Programming Interfaces*

- Selecting the **Use Application User Id for Auditing** option,see the Data Administration Help

Audit archiving

Before you implement auditing, it is wise to form a strategy that includes, among other things, decisions about what to do with the audit data over the long term. When you audit, the audit data initially is recorded in the database being audited. Over time, the audit data might begin to occupy more disk space than is desirable, and at that point you can move the audit data to another, more long-term storage container.

It is, however, your choice whether or not to move the audit data. If the database size is not an issue, you might not want to move the data; however, the longer the audit data resides in a database along with the application data, the higher the risk of data corruption. A best practice approach to the issue of audit data storage is to move the audit data to a database designated to hold the data. In this way, you can identify a designated storage database that has the size you require and establish security on the database such that only those users whose job it is to manage and report on the data have access to it.

Typically, the best audit data security solution involves moving the active databases audit data records from the production server systems to an OpenEdge database that holds only audit data records for long-term storage. Security is enhanced when you place very restrictive access controls on the database to protect the audit data it contains.

When an OpenEdge database is used for long-term data storage, the expectation is that only audit data, and no other application data, resides in the database.

An authenticated user with audit data archive privileges can use the Audit Archive utility to truncate online audit data by moving the audit data to the systems trash basket. The audit data archiver can also archive the data to a secure binary flat file for more convenient offline storage. A MAC data integrity seal is applied to the file containing the audit data so that the files contents can be validated **before** audit data is physically loaded into an OpenEdge databases audit tables.

Caution: The audit data archiver not only archives audit data but can also delete it. Therefore, assigning the audit data archiver privilege to a user is a decision not to be taken lightly.

Archiving audit data

When an audit data archiver uses the Audit Archive utility to move audit data, the following steps occur:

1. The current authenticated user is validated as an audit data archiver. This validation requires that the user authenticate to a user authentication system that is trusted by the OpenEdge database. The initial trusted authentication system is OpenEdge, which is implemented using the OpenEdge databases `_User` table. When no additional user authentication systems are configured for the database, an alternate method is to use the user ID of the audit administrators privilege and use the database MAC key phrase as the account password.
2. The utility establishes a time stamp marker as being the point in time where all older audit data records are selected.
3. The utility creates a new audit data record with the run-time parameters that will be used to perform the archive operation.
4. The utility archives the audit data records from the audit data tables to a secure flat file, from which they can be loaded to a long-term storage database.

If the audit record is an archival event, the utility retains the record in the source. If the record is not an archival event, the utility deletes the audit data record from the source.

For more information, see the section about audit archiving in *OpenEdge Data Management: Database Administration*.

Assigning a database a unique identifier

Each OpenEdge database that is auditing-enabled must have a unique identifier. This unique identifier of the current instance of the database is known as the GUID, or the DB Identifier. The DB Identifier is converted from a uniquely universal (UUID) value, which is a 16-byte optional value.

Assigning each database a unique ID allows you to tell, when you look at the collected audit data, in which instance of a database the auditable event occurred. This is especially important when you have a designated database for long-term storage, and you have moved audit data from two or more databases to that storage location.

For more information, see the Data Administration Help or the Data Dictionary Help.

9

Configuring OpenEdge Auditing

This chapter is intended to help you and your customer production sites get up and running with auditing.

For details, see the following topics:

- Getting started
- Setting up OpenEdge auditing context
- Managing auditing context
- Migrating an existing application to use auditing

Getting started

Getting started with auditing requires the following:

1. Enabling an OpenEdge database for auditing
2. Connecting to the database and assigning audit-related privileges
3. Using Audit Policy Maintenance for policy creation
4. Optionally supplying application context or application events

Enabling an OpenEdge database for auditing

When enabling a database for auditing, you must specify a **Type II** database area for the audit tables. Do not place the auditing tables in the schema area. It is best to create a new, separate Type II area exclusively for the auditing data tables and another new, separate Type II area for the audit table indexes, particularly if you anticipate generating large volumes of audit data. Creating the separate areas might provide improved performance.

For example, you might create the following structure file (which you then add by using PROSTRCT ADD):

```
#
# Add audit areas
#
d "Audit Data Area":13,32;512 . f 512000
d "Audit Data Area":13,32;512 .
#
d "Audit Index Area":14,32;512 . f 512000
d "Audit Index Area":14,32;512 .
```

You can also choose to deactivate the nonunique auditing indexes. This could provide you with a performance boost. Deactivated indexes can be activated at a later time using PROUTIL IDXBUILD.

For more information about Type II areas, see *OpenEdge Getting Started: Database Essentials*.

To enable a database for auditing:

1. Create a database structure file (`.st`) that defines a Type II area for your audit tables and another Type II area for your audit indexes. Be sure to:
   a) Give the new structure file a different name from the existing structure file for the database, so the existing one is not overwritten.
   b) Assign one of the following values to the blocks Per Cluster token when defining the audit table area and audit index area to create the areas as Type II: `8`, `64`, or `512`.

      In the example structure file shown, the blocks Per Cluster token value used is `512` (marked in bold).

2. Add the areas to your database using PROSTRCT ADD. Then use PROSTRCT LIST to generate a new structure file that contains both the original structure and the newly added audit data and audit index areas.

   For more information about creating or adding to an existing structure file and using PROSTRCT, see *OpenEdge Data Management: Database Administration*.

3. Upgrade the clients and the database to the current OpenEdge release (10.1 or later).
4. Enable auditing on the database, as shown:

   ```
   proutil <database-name> -C enableauditing area <"area-name"> [indexarea
   "area-name2"] [deactivateidx]
   ```

   When you do this, the audit tables, which are hidden, are created in the database.

5. Connect to the database as the database administrator.
6. Use Data Administration or the Data Dictionary (character version) to:

a) Set up any internal accounts (`_User`) necessary to define audit administrators and audit data archivers

b) Set up audit security

c) Create a database pass phrase so you can verify that audit data has not been tampered with

For more information, see the Data Administration Help or *OpenEdge Development: Basic Database Tools*.

7. Run Audit Policy Maintenance and load and enable the preconfigured OpenEdge database audit policies (if needed). For more information, see the Audit Policy Maintenance Help.

   For a description of the preconfigured policies, see Preconfigured Audit Policies on page 235.

You can now connect to the database and assign audit-related privileges.

Connecting to the database and assigning audit-related privileges

Once you enable a database for auditing, connect to the database as the database administrator. Then use Data Administration or the Data Dictionary (character version) to:

1. Set up any internal accounts (`_User`) necessary to define audit administrators and audit data archivers.
2. Set up audit security.
3. Create a database pass phrase so you can make sure audit data has not been tampered with.

For more information, see the Data Administration Help or *OpenEdge Development: Basic Database Tools*.

You can now import the preconfigured audit policies provided by OpenEdge or create your own audit policies using Audit Policy Maintenance.

Using Audit Policy Maintenance for policy creation

You use Audit Policy Maintenance to:

1. Connect to an audit-enabled database.
2. Import the preconfigured OpenEdge policies or create your own audit policy or policies and enable audit events.
3. Check for policy conflicts and resolve any you find.
4. Commit the policy settings or changes to the database.
5. Activate the policy.

For more information, see the Audit Policy Maintenance Help.

Setting up OpenEdge auditing context

OpenEdge auditing supports recording events for OpenEdge database record operations as well as OpenEdge application events. These event records provide enough information about the record event itself but lack the contextual information necessary to relate them to the environment in which they occurred.

For instance, it is useful to know when and by whom a database record was changed. However, knowing where it was changed, how it was changed, and what else changed at the same time is required to put that record change into the proper perspective. Without that perspective, it is most often the case that you cannot determine whether the change was legitimate or not, nor will you be able to drill down into the reasons for the change.

For these reasons, OpenEdge auditing provides a number of features that allow contextual information to be associated (bound) to individual audit event records. The types of contextual information that can be associated include any combination of the following:

- Database transactions
- Client login sessions
- Application context
- Application event groups

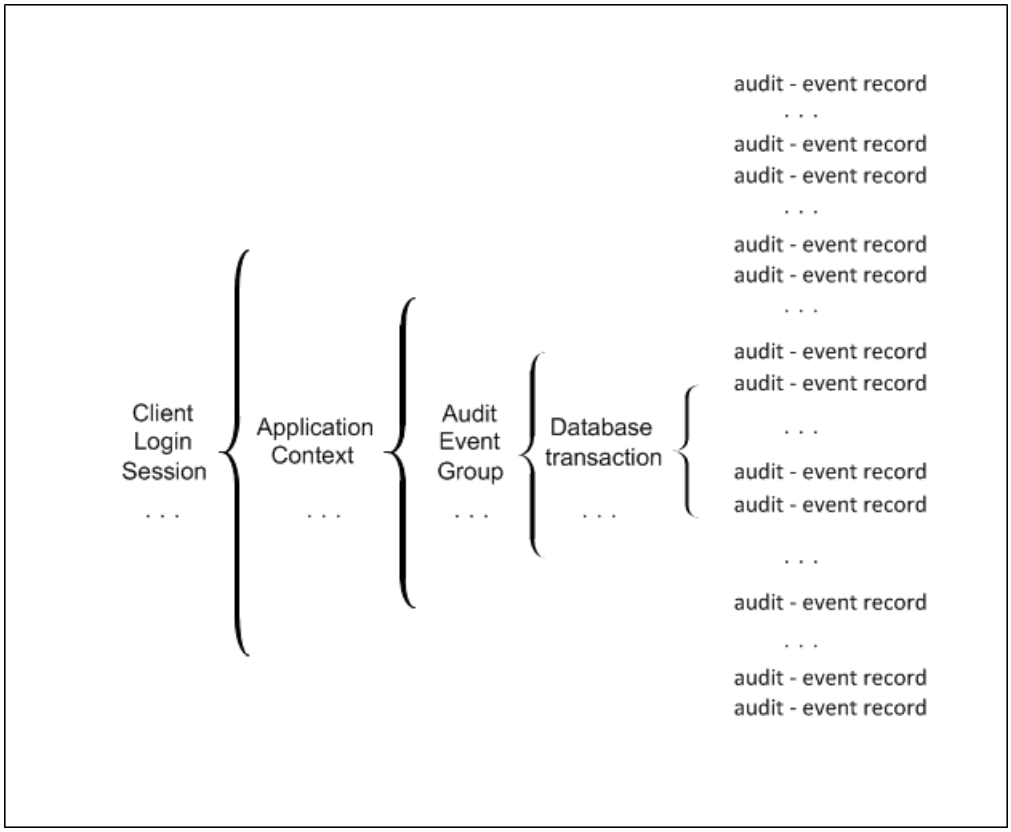
The sections that follow describe the functionality, use, and management of the OpenEdge auditing application context features.

Auditing context architecture

One aspect of applying context to auditing is determining what depth, or how many levels, of context to apply. As current-day applications are far more complex than those of years past, some amount of support for multiple levels of context must be available.

OpenEdge auditing supplies four levels of context information. The following figure illustrates the various auditing context levels available within a running OpenEdge application. Keep in mind that not all context features must be applied to every auditing situation.

Figure 5: Auditing context levels



Database transaction context

The database transaction context identifies all audit events that are related to a single clients database transaction. The database transaction contexts scope is indirectly controlled by the client applications database transaction. When the application starts a database transaction, all auditing records will be associated with that transaction until the application ends the transaction. This happens automatically within the database language clients and does not require additional application code.

Audit-event group

The next outer layer is the *audit-event group*. This is an auditing context whose scope has a beginning and an ending controlled by the application developer. The developer adds language statements to the application that identify exactly where and when to start an audit event group and where and when to end the audit event group. During that time, all recorded audit events will be associated with that audit event group context. Any one audit event group context can encompass zero or more database transaction contexts, and only one audit event group context can be active at one time within an application.

Application context

The next outer layer is the *application context*. This is also an auditing context whose scope has a beginning and an ending controlled by the application developer. The developer adds language statements to the application that identify exactly where and when to start an application context and where and when to end the application context. During that time, all recorded audit events will be associated with that application context. Any one application context can encompass zero or more audit event group contexts, and only one application context can be active at one time within an application.

User login session context

The outermost layer is the *user login session context*. This auditing context reflects a single clients user login session. The scope of this auditing context is indirectly controlled by the OpenEdge application developer through the use of the ABL client-principal object.

Each instance of a client-principal object represents exactly one user login session. When a client-principal object is finalized to signify the successful authentication of the user, the client session context is automatically started. When a client-principal object is ended to signify a user logout, the client session context is automatically ended. During the time that the users login session is set as the current user in the application, all the audit events recorded will be associated with that client session context. The application can have only one active client-principal object set as the current application user at any one point in time.

The illustrations provided here show a typical nesting and usage of auditing contexts. However, it is an illustration provided only in the interest of clarity. Because the application developer controls the scope of the contexts, either directly or indirectly, the relationships can be of any design. For instance, you can choose to nest client session context within an application context; or you might choose to reverse the roles of the audit event group and application contexts. Audit event groups are not necessarily within the application context; both are independent and can be the outer bracket.

The only relationship that **cannot** be changed is that the transaction context must always remain the innermost of all the auditing contexts. The behavior of the transaction context is under the complete control of the OpenEdge clients, and, therefore, does not require developer intervention to be consistently implemented.

Recording auditing context information

The actual information that can be recorded by the various types of auditing context differs. In all cases except the database transaction context, the application developer is in control of certain information fields; OpenEdge controls the remainder.

The application context and audit event group require an event-context field of character information that also serves as an index to locate certain types of contexts for reporting. Optionally, two additional fields of character information, up to a maximum of 10,000 bytes per field, can be supplied. Each time an application context or audit event group context is begun, a single auditing event record is written to the OpenEdge databases `_aud-audit-data` table. The application-supplied information is placed into the record with OpenEdge filling in the remaining user, date, context, and UUID (the unique identifier assigned to an OpenEdge database when the database is created) primary index information automatically.

The client session context takes its information from the client-principal objects attributes and creates a record in the `_client-session` table within an OpenEdge database. The extended user login information is not always necessary for auditing, so recording this information is optional and controlled through database auditing options.

How auditing context is referenced by audit event records

OpenEdge records application context information in one place and then references that context information from many individual audit records. This approach is much more efficient in terms of I/O, CPU, and disk space resources, but requires additional work when creating reports.

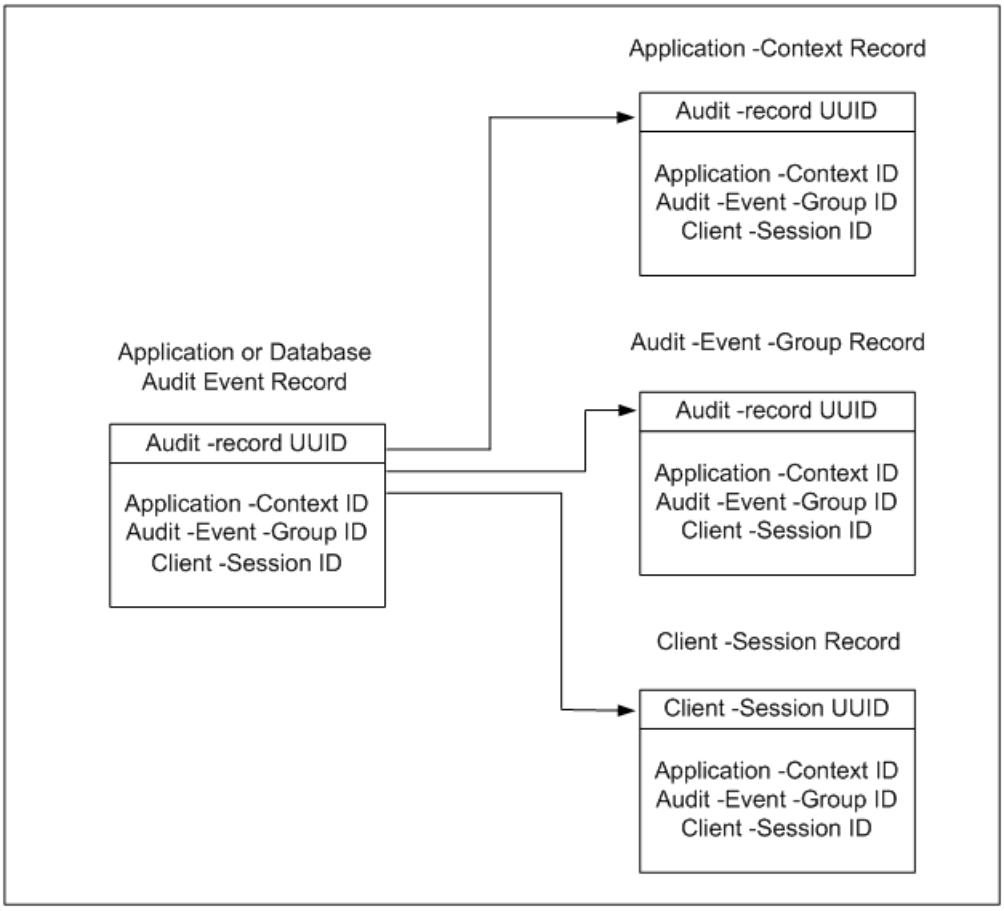
When OpenEdge creates an auditing record, it automatically generates and inserts as the primary index a UUID value that is unique across time and space. The ID values are stored as compressed, character-encoded, Base64 values. The client session, application context, and audit event group records all have one of these ID index fields.

When one of the database clients generates one of these auditing context events, it passes it to each of the connected databases. If the database connection is not audit-enabled, the context event is ignored. Each audit-enabled database connection will record the event according to the events configuration in the databases audit policy. During this process, the primary index ID remains the same regardless of which database the event is recorded into. Even though each databases event record might have a slightly different time stamp, the event record is essentially duplicated. During archival merging operations, these duplicates can be removed without harming the integrity of the auditing data.

Within the auditing record are three indexed, foreign key, character fields that are used to hold the ID index values of client session, application context, and audit event group records. A fourth indexed character field holds the clients database transaction ID.

The following figure shows the relationship among the various auditing records.

Figure 6: Auditing record relationship



Each clients database connection has storage for each of the four auditing context IDs. When the scope of a client session, application context, audit event group, or database transaction context begins, its ID is recorded in each audit-enabled database connection within the client. When the scope of a context ends, the contexts ID is cleared from each audit-enabled database connection so that it is no longer recorded in audit records.

When an audit-enabled database connection generates and writes any auditing record, it will copy the context IDs from the connections storage into the record. This recording of the auditing context ID occurs regardless of whether or not the context event is physically recorded in the database.

Adding auditing context to the application

For details about setting up the application context in the application, see *OpenEdge Development: Programming Interfaces*.

Managing auditing context

OpenEdge auditing is highly configurable through its audit policies. Through audit policies you can, per OpenEdge database connection, designate which audit events will be physically recorded. You can also prevent the application context and audit event group events from being recorded. This does not mean that you lose the recording of the contexts ID in auditing records; it merely indicates that no audit record is generated. The IDs of the contexts will still be recorded so that reporting can make use of the references as a grouping mechanism to find and locate auditing records.

Managing the client session records is slightly different, as recording login sessions are not limited solely to auditing support. In this case, you use Data Administration to control a database option setting. Again, turning off the recording of client session events will result in not writing records. The login session ID is still filled into the auditing record and can be used as a grouping mechanism to locate associated audit records.

The last aspect of managing audit context is the recording of the simple user ID in the auditing records. In some applications, the OpenEdge database connections are under a different user ID than the effective user ID the application uses to operate. In these cases, you want the audit record to record the applications effective user ID, not that of the physical database connection. You control this aspect of auditing through either Data Administration or Data Dictionary by setting a database option that instructs auditing to use the applications user ID instead of the database connections.

Migrating an existing application to use auditing

The following procedure shows how an ABL developer can migrate an existing ABL application, which did not implement its own auditing service, to use OpenEdge auditing.

To migrate an existing ABL application with no auditing to use OpenEdge auditing:

1. Determine if user-defined application auditing events are embedded within your application:
   a. Assign an auditing event ID (above 32000) and context string to each auditing event you embed within your application. You want to define all of the events that any of the end-customers might want to use. Just because an audit event is embedded within the application does not mean that it needs to be triggered at run time.
   b. Change the ABL source code to embed the `AUDIT-EVENT( )` statements into your application, where appropriate.
2. Determine if user-defined application context is linked to database auditing events:
   a. Choose the location, or locations, in your application where you want to begin linking application context to recorded audit events and optionally when to stop. You do not have to record application context on each recorded audit event.
   b. Embed in the ABL source code the `SET-APPL-CONTEXT( )` method for the `AUDIT-CONTROL` system handle.
3. Change your ABL applications initialization to set any customized audit-related settings. (Set up the `AUDITING-DB` alias.)
4. If user-defined application audit events are embedded in the application, export the databases application audit event definitions to an intermediate XML file for loading at the installation site.

Once the application is enabled for auditing, the production installation at the end-customer site follows these steps:

1. Install OpenEdge Release 10.1 and the ABL application. If this is an upgrade, ensure that all components are running Release 10.1 or later.
2. Enable auditing in each of the applications databases.
3. Create the user accounts that will be used by auditors and audit administrators in the OpenEdge auditing databases.
4. Define which user accounts will be given audit administrator privileges.
5. If the application has incorporated user-defined application audit events, import the user-defined auditing definitions into the databases used to store the applications auditing configuration.

6. Configure the database auditing necessary for the end-customer. Using Audit Policy Management or an application-supplied auditing administration utility, configure each database your application uses.
7. If the application has incorporated user-defined application audit events, configure the application auditing necessary for the end-customer through Audit Policy Maintenance or an application-supplied utility.
8. Start the application.

10

Developing an Audit-enabled OpenEdge Application

This chapter describes how to develop an audit-enabled OpenEdge application:

For details, see the following topics:

- Developing an audit-enabled application
- Audit-enabling your OpenEdge application

Developing an audit-enabled application

You begin developing an audit-enabled application by audit-enabling an OpenEdge database. You can then configure auditing in Audit Policy Maintenance and define your application audit events.

Audit-enabling an OpenEdge database

When you audit-enable the database, the PROUTILtool builds the internal tables and provides the OpenEdge internal events used by auditing. Since you have not yet defined any table or field events, however, no audit information is recorded.

For the specific steps you use to enable auditing (including establishing separate audit data and audit index Type II areas in the databases structure file), see Getting started on page 117.

Defining application audit events

You use Audit Policy Maintenance to identify the application-defined audit events you want to use in your application. You supply a unique event ID number (32000 or above), a description, and a type name. You can reuse the OpenEdge type names, or you can create your own type names. You can use the type names to categorize your events, which allow you to group like events together for reporting ease.

There is no single best way to define an application audit event or determine where to place events in your application. In general, one application audit event describes a single application entity, object, process, or task that is auditable but has no related database record operation. For example, use a separate event for a sales order, inventory pull-ticket, or a Web-service method extraction.

For information about application context and audit event groups, see Setting up OpenEdge auditing context on page 119.

Audit-enabling your OpenEdge application

The process of implementing auditing starts with application development. Whether you have an existing application, a new application, or have not yet implemented your own auditing solution, it all starts in the same place. How effective and valuable your application auditing will be at the production site lies in balancing application changes and functionality.

As an application developer, you must do certain things to audit-enable your OpenEdge application. Specifically, you must:

- Document your applications tables and fields so that audit policies can be generated at the production site
- Document the application-defined events that are coded within your application so that they can be configured at the production site
- Supply an application event definition file for all of the application-defined events used within your application so that those events are registered and can be enabled at the production site
- Write ABL and SQL report queries for the tables, fields, and application events contained within your application

Implementing additional auditing options

Auditing has a number of options that you as the application developer can choose to implement in your ABL product. If you implement changes in your application, those options will be available at the production site. If you do not implement the changes, the production site will not have this auditing optional feature. This section lists those development-time options.

If your auditing needs are nonexistent or the OpenEdge databases record auditing is all that is necessary (with no execution context, and if the user who connected to the database is the user to record in the audit log), you do not need to enable auditing for your application.

The databases record provides the following support, without you making any application changes:

- Support of `CREATE`, `UPDATE`, and `DELETE` record operations
- Support for identifying which record was affected
- Support for identifying which record fields were affected
- Support for recording field values at the time the record operation took place
- Identifying which database transaction the record operation was part of

- Identifying which sequence record operations occurred within the transaction
- The date, time, and time zone of the record operation
- The user who made the change, provided the user is either the blank user or the one whose account is in the OpenEdge databases `_User` table, and that account was authenticated through startup/connect `-U` and `-P` options and/or the ABL `SETUSERID` function to the `_User` table

If you do not make application changes to support auditing, you will not have any application execution context available to know which application performed the record operation (SQL or ABL), who the user was (if the ABL application uses its own user accounts), or any other details about the environment in which the record operation was performed.

Supplying OpenEdge application context information

By modifying your application to include application context, you can attach to the audit records information such as:

- **Which application was involved** — This information is necessary if your product has multiple execution components or your product will be part of a partnership with other Progress Software Corporation application vendors.
- **Checkpoint locations within the applications execution** — The checkpoint locations identify the origin of a record operation.

Supporting user accounts outside of the _User table

If you do not want to support populating the `_User` table with all of your applications user accounts and authenticating the application as one of those users, you must change your database connection process to one of the following:

- If necessary, change your database connection process to one of the following:
  - The blank user (with blank user access denied to all)
  - A single default user account, which is used by all application connections, in the `_User` table

  These user IDs can remain for the lifetime of the OpenEdge session or can be replaced by an external user account ID as soon as the connection is made.
- Refer to *OpenEdge Getting Started: Identity Management* for information about establishing an OpenEdge applications current user ID.

Using a dedicated OpenEdge database for auditing

You can store application event data in any audit-enabled database on which the event is enabled. You can use a single storage location or a combination of storage locations; you can also connect directly to a long-term storage area, which eliminates the need to perform the archive step in storing the data.

Setting up READ auditing

If your application auditing requires knowing when certain records are read and presented to a user, you must change your application to:

1. Define an application-defined `READ` audit event in the `_aud-event` table. (Choose an event equal to or greater than 32000.)
2. Define the data formats for the audit records context (indexed) and detail fields. The context field format can be used to index on and find records for reporting. It is recommended that you mimic the OpenEdge database event default field value recording, which streams the field values.
3. Add code to your application where the record read operations occur, and execute the `LOG-AUDIT-EVENT` method (for the `AUDIT-CONTROL` system handle) to record the application-defined event.

   For more information about the method and handle, see *OpenEdge Development: ABL Reference*.
4. Create a `.ad` file to go with your applications deployment to load the `_aud-event` table definitions you have made into the production database instance.

Using a custom audit data/policy archive tool

There are a number of reasons to write your own OpenEdge archival tool. For example:

- To import audit data collected in an older, external system
- To add extra filters to allow more selectivity regarding which audit records are moved or destroyed

When you write such a tool, you have to be very careful, and you might have to certify your archive tool to government auditors. The key to the archival process is that it requires an authenticated user to have the audit data archiver privilege. Only then will the OpenEdge application have `CREATE` and `DELETE` record access to the auditing data tables.

For more information, see *OpenEdge Development: Programming Interfaces*.

Note: `UPDATE` is denied to everyone, including audit administrators, for audit data tables.

Bootstrapping the audit administrator user

To get auditing running, you must set up security, deciding if you are going to have a separation of duty as far as auditing privileges are concerned. Will the security and database administrators manage auditing, or should it be a separate task?

If the database administrators are handling the auditing tasks, the DBAs will want to grant themselves audit data archiver privileges. If the DBAs are not handling auditing and they want to abdicate responsibility to a primary and a secondary audit administrator, those audit administrators can set up the audit reader and audit data archiver permissions for other users.

It is recommended that you have an audit data archiver account separate from the audit administrator account. The audit data archiver privilege can prove potentially dangerous if entrusted to the wrong hands, as the audit data archiver can delete records or create misleading records, thereby creating refutable data.

For more information, see Audit Security on page 103.

Creating audit policy and report templates

Your database application will have many databases, tables, and fields. Without specific knowledge of the schema, the production site will not be able to create auditing policies and run reports. As the developer, you have the knowledge of the schema and can provide audit policy templates and report queries to the production site. This goes along with the schemas documentation. The production site can then copy, customize, and activate the templates with far less work and ramp-up time.

One thing you do not want to do is turn **everything** on to be audited and then let the production site report only on events of their choosing. Such an approach would likely consume much storage space and increase greatly the run-time overhead. If you know for certain that specific events do not need to be audited, be sure not to identify them as audit events.

The same situation applies to report queries. From templates, the production site can do less work and be running faster if it can take advantage of the developers knowledge.

Supporting custom audit policy tools

When a large, complex application is involved, it can be impractical to expect a production site to manage audit policies for the applications many tables across multiple databases. In such a situation, the developer can customize the audit policy tool by applying knowledge of the application and database schemas to the user interface of the customized policy tool based on shared ABL procedures, thereby greatly simplifying the audit policy configuration and maintenance.

For example, you can create a policy in segments with one segment holding a common configuration for a type of report or reports. The production site can then turn on the segment.

Updating an existing ABL application with auditing

If your application already has auditing support, you can migrate the application (provided you have audit data archiver privileges) to use the new OpenEdge auditing by doing the following tasks:

1. Migrate existing data into the OpenEdge auditing schema format using a designated ABL application run by an audit data archiver.

   This includes adding to the application any code required to support auditing, creating a policy, identifying the events to audit, committing the changes, and activating the policy. Additionally, the schema trigger settings must be removed.

2. Migrate your applications trigger code to remove the record-level information and form the application-level information to fit into the ABL language statements and auditing record field formats.

Audit-enabling your SQL application

Custom ODBC or JDBC can add auditing SQL statements to add application context and application-defined events, just like an ABL application. For more information, see *OpenEdge Data Management: SQL Development*.

11

Deploying an Audit-enabled OpenEdge Application

This chapter describes how to deploy an audit-enabled application.

For details, see the following topics:

- Overview
- Configuring auditing at the production site
- Applying audit policy for production systems
- Writing custom audit reports
- Preparing to deploy your audit-enabled application
- Upgrading an existing application to use auditing

Overview

Many of the decisions regarding auditing configuration will be determined at the production site. This includes configuring what will be audited, where the audit data will be stored, how audit policies will be created and maintained, how the audit data will be maintained, and how the audit data will be reported.

This section describes the tasks that are likely to occur at the production site; the information is addressed to the application developer, who can pass it along to customers.

Developing the company's audit policy

It is possible that two companies that use the same application will not configure and manage auditing in the same way. Because there will often be something that will make a single solution unacceptable, OpenEdge auditing supports many configurations.

Before implementing auditing, the first thing a production site should do is create its auditing strategy. In creating the strategy, the site should consider the information provided in the following sections and how it applies to the sites unique situation.

Configuring long-term audit data storage

How long will you need the audit data to remain online so that you can run necessary reports? The length of time, combined with how much audit data you will be accumulating, will allow you to plan for how much physical disk capacity is needed and whether you will require multiple physical databases in which to store the audit data. Any long-term storage database should be tuned to store large amounts of data and be very secure, since auditing can accumulate data at a very fast rate.

Audit-enabling the applications OpenEdge databases

When your application supports multiple databases, not all of them must be audit-enabled. If a database has no auditable tables in it, you do not have to slow it down by turning auditing on. If the application vendor has not already preconfigured the database for auditing, see Audit-enabling an OpenEdge database on page 127 details about getting started.

Configuring auditing at the production site

Configuring auditing at the production site involves four overall steps:

1. Setting up audit administration
2. Loading audit event definitions
3. Loading predefined audit policies
4. Setting up audit data access

Setting up audit administration

Production sites might choose to combine the duties of administering audit data with that of administering the entire database, or they might choose to separate the responsibilities into two distinct functions. By default, the OpenEdge database provides the database administrator with rights to be the audit administrator.

If you prefer not to use the default and instead want to separate the functions, do the following:

1. Designate the first user account that will be an audit administrator.
2. Using Data Administration, grant that user account the audit administrator privilege and the right to grant it to other users. For more information, see the Data Administration Help.

   Note: It is recommended that you have an alternate audit administrator in place to handle any unforeseen emergency situations.

3. Disconnect from the database and then authenticate as the designated audit administrator. You can now grant auditing privileges to other users and begin configuring the auditing policy for the database.

Loading audit event definitions

When your OpenEdge application supports application-defined auditing events, it must register (that is, load/import) them in each auditing-enabled OpenEdge database. If your vendor has not already preconfigured this into the databases for which you have enabled auditing, you must locate the audit policy configuration `.ad` or `.xml` files that should have shipped with the product. You will use Data Administration to load the audit event definitions for `.ad` files; you can use either Audit Policy Maintenance or Data Administration to load `.xml` file definitions.

Loading predefined audit policies

OpenEdge database schemas can be very complex and make the task of knowing what to configure to meet your auditing goals a challenge. For this reason, your applications vendor might choose to supply preconfigured audit policies that you can simply load, modify to meet your needs, and activate. The loading of these policies would follow the same approach described in Loading audit event definitions on page 135. After the policy templates are loaded, you can use the Audit Policy Maintenance tool or the vendors audit policy tool to perform the policy changes and activation to meet your sites auditing requirements.

Setting up audit data access

You will normally have people whose job it is to access audit data, create reports, and analyze those reports. Most of them do not, and should not, have the privileges to change audit policy or maintain the audit data. That is why you can grant audit read privileges to individual user accounts. Determine which individuals require this ability and have the audit administrator grant that privilege to those user accounts.

Applying audit policy for production systems

Each audit-enabled OpenEdge database can contain any number of audit policies, and any combination of policies can be designated to be active at run time. Audit policies are ignored until they have been activated; additionally, audit policy changes are ignored until the audit administrator is satisfied with the changes and commits them to the run-time system. Generally, the flow of operations is:

1. Load audit policies.
2. Customize policies.
3. Enable policies.
4. Check for conflicts and errors.
5. Update/enable/disable policies.
6. Notify the OpenEdge run time to use the newly selected set of enabled policies.
7. Monitor run-time audit behavior.
8. Return to Step 2, if necessary.

Writing custom audit reports

It can be difficult for an application vendor to predict all the different ways that any one production site might want to report on its customized set of application data records. Therefore, the production site might have to write its own reports using either ABL or SQL.

The general process is as follows:

1. If the vendor has supplied template queries, determine whether one of those is close to what is needed.
2. Based on the audit data being collected through the audit policies in effect, use the vendor or OpenEdge schema information to create or customize the database query. For details about the OpenEdge schema, see Audit Data Tables on page 201.
3. Write the query and data presentation.
4. Test and debug.

Note: Be aware that if the report is for a government auditor, you might have to certify the truthfulness and trustworthiness of your report generator.

For more information about writing custom audit reports, see *OpenEdge Development: Programming Interfaces*.

Preparing to deploy your audit-enabled application

To prepare to deploy an audit-enabled application, follow these general steps:

1. Use the PROUTIL command to create the new OpenEdge audit-enabled database and define your applications schema.
2. Use Audit Policy Maintenance to import any predefined auditing policies for each of the audit-enabled databases.
3. Optionally, use Audit Policy Maintenance to import your application-defined auditing event definitions into the OpenEdge databases.
4. Use Audit Policy Maintenance to configure one or more audit policies to satisfy the end-customers needs.
5. Use Audit Policy Maintenance to configure one or more audit policies that your databases will employ at run time.
6. Use Data Administration to configure your databases auditing security settings. For more information, see the Data Administration Help or *OpenEdge Development: Basic Database Tools*.
7. Optionally, use the command-line tools to automate archiving from short-term to long-term storage.

Upgrading an existing application to use auditing

If you have an application that does not have its own auditing functionality, you can upgrade the application to use OpenEdge auditing.

To upgrade the application to use OpenEdge auditing, follow these general steps:

1. Use PROUTIL to upgrade your OpenEdge database products to OpenEdge Release 10.1.
2. Use PROUTIL to enable and configure your OpenEdge database's auditing tables.
3. Use Audit Policy Maintenance to import any predefined auditing policies for each of the auditing enabled databases.
4. Optionally, use Audit Policy Maintenance to import your application-defined auditing event definitions into the OpenEdge databases.
5. Use Audit Policy Maintenance to configure one or more audit policies to satisfy the end-customers needs.
6. Use Audit Policy Maintenance to configure one or more audit policies that your databases will employ at run time.

7. Use Data Administration to configure your databases auditing security settings. For more information, see the Data Administration Help.
8. Optionally, use the command-line tools to automate archiving from short-term to long-term storage.
9. Upgrade the database clients to OpenEdge Release 10.1.

Changes in audit policy settings used by the OpenEdge database engine take effect immediately. Changes to audit policy settings for ABL or SQL database clients are picked up on the next new connection.

You can archive and/or record application audit data to any local- or network-accessible OpenEdge database.

12

Maintaining Audit Data

An important part of developing and implementing a successful auditing solution is maintaining the generated audit data.

For details, see the following topics:

- Common audit data maintenance tasks
- Auditing performance
- Run-time audit maintenance

Common audit data maintenance tasks

Audit data life cycle management is an essential part of any auditing solution. When a site implements auditing, it is anticipated that large amounts of data will be collected, stored, and retrieved over a period of several years.

Backing up and restoring an audit-enabled database

The database administrator uses the same tools and processes when backing up and restoring an audit-enabled database as are used for any other OpenEdge database that is not enabled for auditing.

Archiving and loading audit tables

Only a user with audit data archiver permission can audit archive and audit load audit tables, using the Audit Archive utility supplied by OpenEdge (or a self-designed archive tool). If an audit data archive file is altered in any way, you cannot load it. The archive and audit load operations themselves are audited to track the audit data archiver who performed the archive and archive load operations.

Note that the audit administrator can read but not archive or audit load the audit data. The audit data archiver must maintain the audit data.

Copying an audit-enabled database

The database administrator uses the same database utilities to copy an audit-enabled database as would be used for a database that does not have auditing enabled. The copy operation itself is also audited to track the database administrator who performed the copy.

If the database is being copied to create a new database instance, the database administrator would also add the command-line options necessary to initialize the new database with its own identity (UUID). If the copy is read-only, no new UUID is necessary.

Recovering audit data

In the event of a catastrophic failure involving audit data, the database administrator uses the same database utilities and processes as when recovering nonaudit data.

Modifying the indexes generated for OpenEdge audit tables

You can use PROUTIL to perform the standard index rebuild process to modify which audit data indexes are enabled and also to rebuild the indexes themselves. You must have audit data archiver permissions to delete an auditing data record associated with a bad index.

Modifying the audit table storage area

Only an audit data archiver can truncate an audit table storage area.

Disabling auditing

To disable auditing only temporarily, the audit administrator can disable all policies through Audit Policy Maintenance.

To turn off auditing **permanently** (with no intention of ever re-enabling it), the audit administrator can use the PROUTIL utility. No existing data will be lost. For more information about disabling auditing, see *OpenEdge Data Management: Database Administration*.

Note: Pre-Release 10.1 OpenEdge clients are blocked from access to the database until all the data in the audit tables has been deleted.

Handling long-term storage growth

You can archive audit data using either the Audit Archive utility provided by OpenEdge or an archive tool you have developed. In both cases, audit data archiver privileges are required.

If your long-term storage area grows too large, you can do any of the following (again, provided you have the requisite permissions):

- Use the archive tool to truncate the data.
- Use the archive tool to move the oldest audit data to a new OpenEdge database and then move that database to offline storage.

- Archive the audit data tables and move the files to long-term storage. Then, use the archive tool to truncate the online audit data.
- Develop a full set of audit reports for the oldest audit data and archive the reports. Then, use the archive tool to truncate the online audit data.

Returning audit data from offline storage

The method you use to return data from offline storage depends on how you moved the data offline. If you performed an audit data table archive, you can load the table data into an accessible online OpenEdge database. Since you will likely want to remove the audit data again once you have generated reports, be sure to make the location somewhere that is safely and easily removed from online status.

You must have audit data archiver permissions to audit archive and audit load the audit data.

If you store a backup copy of an OpenEdge database, you can restore the database to an online state and add it to the list of audit data locations that the audit report generator accesses.

Ensuring audit data integrity

You can configure auditing security to turn on data integrity checking. You can use the archive utility to perform a data integrity check on demand. The data integrity checking reads each audit data record and validates that the data has not been modified. Any failures are reported to the user of the utility.

Auditing performance

When you set up the audit configuration through the audit policies, be sure that you are auditing only those events you want to record. The more audit events you set up, the higher the amount of overhead time auditing consumes, taking away from application data operations.

You can ask your database administrator to use the PROUTIL utility to disable building of all the optional indexes on the audit data tables. You would normally do this if you are not going to run audit reports against them.

This practice is recommended when you regularly archive audit data to a database dedicated to holding long-term audit data. Enable full index support only on the dedicated, long-term audit database.

You can also configure auditing to store old and new field values, no values, new values only, or old values only. The less you store, the more efficient the auditing performance is.

Run-time audit maintenance

Once OpenEdge auditing is running, it must be monitored and maintained. The monitoring and maintenance are critical to keeping applications running. It will be easy to fill up an application database or file system with audit records and thereby shut down the application. This section describes some of the daily auditing maintenance tasks.

Monitoring your databases health

Auditing data can grow at a rapid rate, even if you are careful to record only what is absolutely necessary. Because OpenEdge databases and the disks they reside on do not have infinite resources, you must have a good monitor running that has an alerting feature. Should the database begin filling at an abnormal rate, this monitor must trigger an alarm so that the situation can be investigated and corrected.

The monitoring process also serves as a high-water mark to let you know that you must begin planning an off-line archival of the audit data to make room.

Querying audit data

For details about querying or reporting on data in the audit tables, see Querying and Reporting on Audit Data on page 143.

13

Querying and Reporting on Audit Data

This chapter describes some of the intricacies of auditing data and factors to be aware of as you query audit data.

For details, see the following topics:

- Overview
- Audit data querying and reporting
- Reporting on audit data with prefiltered and custom reports
- Generating a custom filtered report

Overview

You can query the audit data and also generate reports from its contents. To create system-generated or customized reports for internal OpenEdge events, you can use the reports provided by OpenEdge; for more information, see Reporting on audit data with prefiltered and custom reports on page 156.

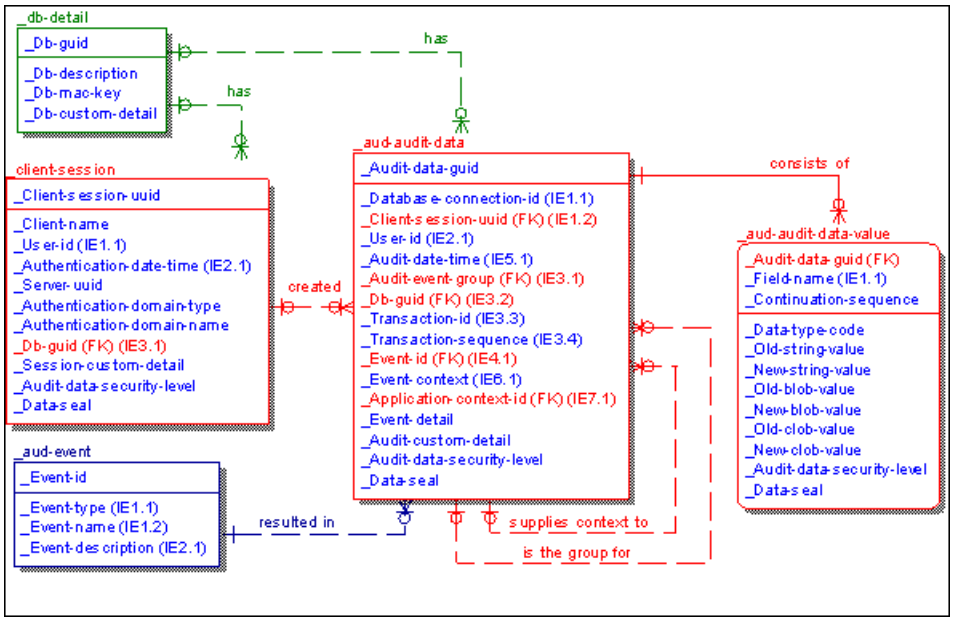
You can also create audit data reports by using your own general-purpose report-generation tool. If you do choose to create your own reports, an understanding of the audit data schema is necessary.

Note: The contents of this chapter assume some knowledge of the OpenEdge auditing architecture and functionality described earlier in this document.

Audit data schema overview

The following figure shows an entity relationship diagram for the audit data and supporting tables and provides a brief description of each from a reporting perspective.

Figure 7: Audit tables schema



Note the following points:

- The primary table is `_aud-audit-data`, whichholds the details of the event that was audited, such as who did it and when, for example. It may also contain a delimited list of modified field values depending on the audit policy level settings.
- The `_aud-audit-data-value` table optionally stores additional details for modified fields showing the new and, optionally, the old values. The level of detail recorded (for example, simply the fact that something happened, the new value of modified fields, both old and new values, or a stream of an audited streams values) is driven by the table policy. Records are created in this child table only if the policy is set to record one record per field.
- The `_client-session` table optionally records additional session information such as the authentication domain and client name.
- The `_db-detail` table provides a description of the database for which the audit data exists, as well as the passkey used to optionally seal the audit data records.
- The `_aud-event` table provides additional audit event information that is useful for reporting, rather than simply showing the integer event ID. The event definition itself determines the structure of the information in the other tables.
- A Foreign Key is denoted by the abbreviation FK.
- The fields above the line are the Primary Key fields, in order.
- **IEx.y** denotes an Inversion Entry (nonunique alternate index) key with the first number (x) being the index number and the second number (y) being the position of the field within the index.

Audit data querying and reporting

The following sections describe considerations related to querying and reporting on audit data.

Reporting committed data only

When reporting on audit data, you should be sure that only committed data is included in the report. Due to the way ABL and SQL clients and the database work, it is possible that the database might contain interim audit data that could be rolled back in the event of a failed transaction. Therefore, if you are reporting against an operational database that has auditing enabled and active, it is important that the audit data be read `SHARE-LOCK` to avoid reads on uncommitted data resulting in inaccurate audit reports.

Internationalization considerations

Modified field values are not stored in the audit tables in their native data type; they are stored as character strings. In order to ensure consistent formatting of the data, the values are always stored in the database in American format (`SESSION:DATE-FORMAT` = "mdy" and `SESSION:NUMERIC-FORMAT` = "American"), regardless of the client settings used to record the audit data.

If the data must be reported in a different format or restored back to its native data type, the session must be temporarily changed to American format, the value converted back to its native data type, and the session restored back to the correct format. This will avoid the occurrence of any type of conversion error.

Additionally, event context stores a delimited list of field values in a character field, so it also uses an American format in all cases. This must be considered when constructing a query to locate specific audit data using noncharacter identifying fields.

Audit data is always stored in the code page of the database. You can only audit archive and audit load data between databases with the same code page.

Reporting on a central archive database

By using an audit archive utility, you can source audit data from multiple short-term application databases and aggregate it into a central audit archive database purposed for long-term storage and security. The originating database is identified by the designated database GUID field `_db-guid`, which uniquely identifies all Release 10.1 databases. The archive utility also copies records in the `_db-detail` table, which contains a record for each database GUID, and provides additional information such as a description of the database for reporting purposes as well as security information.

Any reports on the audit data should join to the `_db-detail` table using the `_db-guid` to show more detail about the database, such as the database description, if available.

Reporting on a single database with multiple GUIDs

If a reporting requirement is to group related data for a specific database, note that it is possible that a single database GUID (`_db-guid`) can change over the lifetime of the database, such as following an update to the database passkey, for example. Therefore, data for a single database could, in fact, require reading multiple database GUID values.

There is no easy way to determine which of the GUID values are for the same database. One possible way is through the structured use of the description field or possibly using the custom detail field on the `_db-detail` record, or alternatively reading the list of GUID values from the originating short-term database if access to this database is available.

Reporting from multiple audit databases

In some scenarios, it might be necessary to report against multiple audit databases, for example, to report on data in both the short-term operational database (application database) and a designated, audit archive database. If this is a requirement, then the reports used should be developed to accommodate extracting data from multiple data sources and aggregating the results at run time. If you are adopting this model, take care to resolve duplicates; it is possible that the same audit data can exist in each of the databases, which would ordinarily be resolved as part of the archive into a central reporting database.

Deactivated audit indexes

When auditing is enabled for a database, it is possible to have some of the audit indexes created as deactivated. This is the preferred approach for the database used to record the audit events, as it will have the least impact on the operational database. In the database used for reporting, however, the audit indexes must be activated for better report generation performance.

Reporting event descriptions

The audit event is recorded in the audit data table as an integer ID to save storage space and also to facilitate logical grouping of events by ID. Reports should join to the `_aud-event` table to display a more meaningful description of the event. The `_event-description` field should be structured in a meaningful way to appear on reports and would be the preferred field to use in the report. Note that for internal OpenEdge-defined events (with event IDs lower than 32000), the event description cannot be updated.

When defining audit events, you should ideally group the event IDs into ranges so that a range of events can be queried to group related information. This grouping approach has been followed as closely as possible for internal OpenEdge-defined audit events and would be a best practice for application events also.

Additionally, for internal audit events, the event name is structured to facilitate logical grouping so that related events can be queried using partial event name searches. Where such a query is required, the query has to start with the `_aud-event` table to first resolve the list of event IDs of interest.

Optional client session information

It is possible that additional client session information has been recorded in the `_client-session` table. Reports on the audit data should, therefore, check for the existence of such data using the `_client-session-uuid` foreign key field. If the data exists, the report can optionally include it. Note that although a value can exist for the client session ID, a detailed client session record (`_client-session`) cannot exist. The client session records are recorded only for authenticated sessions that use the client-principal object along with the `SET-CLIENT( )` or `SET-DB-CLIENT( )` ABL code to assert the identity of the authenticated user.

There is a fair amount of data related to the client session, so it is more likely this would be either reported in a separate section listing the client sessions for the other criteria reported on or reported only in a designated report detailing what happened in a specific session, as identified by the session UUID.

Audit data event context

The event context (`_Event-context` field of the `_aud-audit-data` table) is an indexed field and the primary mechanism for locating what audit data exists for a specific event ID, such as what changes have occurred for the specific customer number. It is extremely important, therefore, that the event context be structured in a useful and consistent manner according to the type of event. If the structure or format of the event context varies for a specific event, it would be difficult to locate records over time—especially since it is not possible to update audit event data to correct the format at a later time. For security reasons, audit data can never be updated, under any circumstances.

Each audit event must have a well-defined and published event context format to facilitate efficient querying. For application events, this is left to the developer to define, and care should be taken to record this consistently. For internal and database events, each event has a predefined structure and format.

For audit events, where no database record operation is involved, such as for internal database PROUTIL utility events, the format of the `_Event-context` field is as follows:

```
[ { T:tenant-name | G:group-name } . ] owner.table-name
```

All database event record operations use the following format:

```
[ { T:tenant-name | G:group-name } . ] owner.table-name
CHR(6)id-fld-val [CHR(7)id-fld-val-n . . . ]
```

`T:`*`tenant-name`*

For a multi-tenant database only, *`tenant-name`* identifies the name of database tenant on which the audited operation occurred.

`G:` *`group-name`*

For a multi-tenant database only, *`group-name`* identifies the name of tenant group on which the audited operation occurred.

`owner`

The owner of the database table on which the audited operation occurred. For ABL, this is always `PUB`. Otherwise, it is the table owner as defined by SQL.

`table-name`

The name of the database table on which the audited operation occurred.

`CHR(6)`

`CHR(7)`

Non-printing ASCII characters with the value of 6 to delimit an initial identifying field value, and the value of 7 to delimit each additional identifying field value (respectively).

id-fld-val

An initial field value that identifies the database record on which the audited operation occurred.

id-fld-val-n

One of zero to any number of additional field values required to identify the database record on which the audited operation occurred.

The *id-fld-val* and *id-fld-val-n* refer to the identifying field values as defined in the table policy records, defaulting to a tables primary key fields.

Note that identifying fields can be used only if they belong to the table being audited; it is not possible to specify a field from some related table to use in place of the foreign key value. This presents limitations in the case where the foreign key is a meaningless object ID, and a GUID and more meaningful data exist only in the related table that the GUID points at. In such cases, the report must have an internal knowledge of how to join to the related table at run time, as well as have access to the database the related data resides in, which might be an issue for historical or deleted data.

An option to work around this is to force audit data to be created for the related table with full detail being recorded of the required field values, and then to retrieve this information from the audit data tables as well, removing the dependency on the original database and schema.

The event context field is a character field, as described, and is limited to 200 characters (because it is indexed). This could also present some limitations for tables with many or large identifying field values, as part of this 200 is taken up by the owner and table name as well as the delimiters. For reporting or querying, this could result in truncated values; give careful consideration as to the identifying field values used.

Note that locating specific audit data for a record requires knowledge of the structure of the event context, and, therefore for database events, knowledge of the identifying fields used—as it is only the value that is recorded—not the field name. The field values will be stored in American format for consistency, so also consider this when constructing queries to locate specific audit data based on the audit context.

It is possible to have multiple active audit policies with potentially conflicting sets of auditing fields for the same table policy. In cases where this occurs, the aggregation of all the fields marked as identifying fields across all the active policies will be used. If multiple fields have conflicting ordinal positions, the order will be unpredictable.

The implication of this is that the format of the context will be unpredictable and different for data in the same table, depending on the active policies. This could get very confusing and make it almost impossible to query against specific audit data for specific records. Avoid this situation at all costs. Unfortunately, it is difficult to prevent; but Audit Policy Maintenance will at least provide a report that highlights the issue. This report should be run to identify conflicts before using the policy. For more information, see the Audit Policy Maintenance Help.

Audit data application context

Application context (`_Application-context-id`) can be used to provide additional information as to the application context under which the audit event data was created, such as the execution context including the call stack that would identify the procedure code that caused the audit event, for example. The use of the application context and what data it contains is driven by the application code using the new statements `AUDIT-CONTROL:SET-APPL-CONTEXT` and `AUDIT-CONTROL:CLEAR-APPL-CONTEXT`. In SQL, the statement `AUDIT SET APPLICATION_CONTEXT` is used to set it to a string or clear it by passing in NULL.

When the application context is set, a complete audit record will be created recording the details of the context against the context event ID 31998. The value of the `_Audit-data-guid` identifying field value for this context audit data record will then be set in the `_Application-context-id` field of all subsequent audit data records until the application context is cleared. It is, therefore, the responsibility of the application to correctly set and clear the context at appropriate points. Be extremely careful to handle errors appropriately and ensure that the resetting of the context does not get skipped due to an error condition.

Where multiple databases are connected and audit-enabled, the application context record will be written to each database that has enabled the recording of the application context event ID through the event policy. The context record will, however, be created identically in each database and conflicts resolved when archiving audit data from multiple databases into a centralized reporting database. Regardless of whether the database has the application context event enabled, the context ID will be set in **all** connected databases, ensuring that the correct ID is set on all audit data records in all databases, until the context is cleared.

In order to report on the details of the application context, a secondary read back into the audit data table will be required, locating the audit data record where the `_Audit-data-guid` value matches the `_Application-context-id` value.

In the context record itself, the fields that could contain data passed in from the application and supported by the API are the following three character fields: `_event-context`, `_event-detail`, and `_Audit-custom-detail`. Other fields, such as the user or date and time, will also be populated automatically as with database events.

Using the `_Application-context-id` field itself makes it possible to read all audit data that resulted from the same application context.

Logically grouping related audit data in an audit event group

Audit event groups (`_Audit-event-group`) provide another means for applications to group related audit data according to specific application requirements, such as to group all audit data related to a specific business entity, task, or workflow, for example. Audit event groups have starting and ending events that can be used to establish a collection of related audit events that could span multiple databases (as each database that has the audit event group enabled through policy will receive the audit event group record).

ABL statements used to create and clear audit event groups are `AUDIT-CONTROL:BEGIN-EVENT-GROUP` and `AUDIT-CONTROL:END-EVENT-GROUP`. These work much the same way as the setting and clearing of application context. In SQL, the statement is `AUDIT SET EVENT_GROUP` passing in a string or NULL to clear it.

When the audit event group is set, a complete audit record will be created recording the details of the audit event group against the audit event group event ID 31999. The value of the `_Audit-data-guid` identifying field value for this event group audit data record will then be set in the `_Audit-event-group` field of all subsequent audit data records until the event group is cleared. It is, therefore, the responsibility of the application to correctly set and clear the audit event group at appropriate points. Be extremely careful to handle errors appropriately, and ensure that the resetting of the event group does not get skipped due to an error condition.

In order to report on the details of the audit event group, a secondary read back into the audit data table will be required, locating the audit data record where the `_Audit-data-guid` value matches the `_Audit-event-group` value.

In the event group record itself, the fields that could contain data passed in from the application and supported through the API are the following three character fields: `_event-context`, `_event-detail`, and `_Audit-custom-detail`. Other fields, such as the user or date and time, for example, will also be populated as normal.

Using the `_Audit-event-group` field itself makes it possible to read all audit data that resulted from the same event group.

Grouping audit data by transaction ID

Whenever audit data is created, the current database transaction ID will be recorded in the `_transaction-id` field and within a transaction, for auditable events, the `_transaction-sequence` will be updated to provide the order in which things happened within the transaction.

This is useful in identifying what occurred within the bounds of a single database transaction. It could likely be used when looking at a specific field value change to see what other information was updated in the same transaction that could have impacted the new value of the field.

Note that transaction IDs can loop back to zero, so take care when reporting by transaction ID across multiple transactions. The date and time should be used with the transaction ID to get a more accurate sequencing of events.

Note also that transaction IDs are not unique to a database.

Grouping audit data by database connection

All database-related audit events will have the database clients connection ID written to the audit data record, making it possible to report on all activity for a specific database connection. Note that in an OpenEdge AppServer model, this could represent activity from multiple users.

Reporting additional audit event details

For application events, these fields can be used to supply additional information relevant to the specific application. Their meaning and use are, therefore, completely application-specific and will likely be linked to the event itself.

For database events, the `_event-detail` may be used to record modified field values depending on the audit policy level and whether a separate record per field is being recorded or not. For more information, see Reporting modified field old/new values on page 150.

Reporting by audit date and time

Note that the date and time recorded in the audit records is the databases date and time rather than the clients date and time, since the database is creating the audit data records itself. Also note that the format of the date and time is American (mdy).

Reporting by user ID

The user ID recorded for the audit data depends on how the user has been authenticated as well as the settings in the database options. The possibilities are the current user who connected to the database (using the `-U` and `-P` connection parameters), the user as set by the `SET-DB-CLIENT` function for a particular database, or the application-wide user set by the `SET-CLIENT( )` method.

Reporting modified field old/new values

The recording of modified old and new field values in the child `_aud-audit-data-value` table is optional and driven by the level settings in the table policy. It is also currently relevant only to database `CREATE`, `UPDATE`, and `DELETE` events.

It is optional whether to record modified field values at all. When modified field values are recorded, using the child table to record each modified value in a separate record is also optional—driven by file and field policy. The default is rather to record a delimited list of modified field values in the parent table `_aud-audit-data` in the `_event-detail` field, as this is generally the better performing option but is harder to query and report against.

One option is not to record any old or new values, in which case no child records will exist, and the only means to determine what the change relates to is by the identifying field values in the event context.

A second option is to record only new, modified values. Determining what the value changed from could require the reading of many prior records until a match is found to modify the same field.

The final option is to record both old and new values to make it easier to report on exactly what the value changed from and to.

Note: The recording of old and new values for BLOBs and CLOBs is not supported.

These settings can be modified for individual fields, perhaps to record only the fact that a record was changed for the majority of fields and to record the actual value change for just a few specific fields. Additionally, specific (large) fields could be set to record modified values in separate child records while other fields are recorded in a delimited string in the parent record. Therefore, determining why and where certain values exist in the audit data tables would require knowledge of the audit policy that was in place at the time. Reports should handle the fact that these detail records might or might not exist.

Given the flexibility permitted with recording modified field values, reports should always check both the parent record `_event-detail` field and the child records in order to gather up all of the modified field values. The simplest approach is to turn the delimited list of modified field values back into separate child records in memory for reporting (using a ProDataSet). Even if the file or field policy is set not to record modified values in separate child records, if the data does not fit into the record, then the default mode is ignored and separate records are recorded for each modified field value. This is another reason why both locations must always be checked, regardless of the policy settings.

Apart from BLOBs and CLOBs, the modified values themselves are stored in a character field using an American format for type conversions. To facilitate casting of the value back to the correct data type and appropriate format, the originating data type of the field is also recorded.

The data type is an integer code with the sample values shown in the following table:

Table 8: Sample data type values

Data type	Sample value
1	Character
2	Date
3	Logical
4	Integer
5	Decimal
7	Recid
8	Raw
11	Memptr
13	Rowid
18	BLOB
19	CLOB
34	Datetime

Data type	Sample value
39	Longchar
40	Datetime-TZ

Because the field name, old value, and new value are all being stored, it is possible that the record limit of 32K could be achieved if the field values are very large. To avoid this, the audit data value record supports *continuation records* through the `_continuation-sequence` field. The purpose of the continuation sequence is, therefore, to facilitate breaking a single audit record value into multiple parts to avoid breaking record limits where the audit data value is too large to fit into a single record.

When reporting is being done on old or new values, take into consideration the fact that the data can be spread over multiple records. Ordinarily, the continuation sequence will be 0. If continuation records exist for the same `_Audit-data-guid` and `_Field-name`, then additional records will exist with a continuation sequence greater than 0, each containing as much of the remaining data as possible.

Where a modified field value is for an array field, the value will be stored `chr(8)` delimited as follows:

```
Format: E[n]:<value1>chr(8)E[n]:<value2>chr(8)
Here is an example:
```

```
E[8]:279.67chr(8)E[11]:0
```

`E[n]` means the *n*th element in the array; the array index starts from 1.

When recording modified field values in the parent record in `_aud-audit-data._event-detail`, the structure of the data is as follows:

```
fieldname + chr(6) + data type code + chr(6) + old value + chr(6) + new value + chr(7)
```

The old value entry is only relevant to update level 13, so in all other cases it will be left blank to ensure a consistent number of entries to simplify reporting.

When modified field values are recorded this way it is not possible to efficiently search for where a specific field changed and to what value. If this is important, then store the field in separate child records, balancing the operational performance overhead against reportability.

Identifying database events

It is important to be able to determine whether an event is a database event in order to know what the format of the `_event-detail` field is and whether to expect any child records in the `_aud-audit-data-value` table.

One way is to check the event definition and rely on the event name ending in `.create` or `.delete` or `.update`. All the internal policies conform to this. This can be checked in code as follows:

```
DEFINE VARIABLE lDbEvent AS LOGICAL NO-UNDO.
IF _event-name <> "":U AND
   (ENTRY(NUM-ENTRIES(_event-name,".":U),_event-name,".":U) = "create":U OR
    ENTRY(NUM-ENTRIES(_event-name,".":U),_event-name,".":U) = "delete":U OR
    ENTRY(NUM-ENTRIES(_event-name,".":U),_event-name,".":U) = "update":U) THEN lDbEvent
 = TRUE.
```

This approach will work only if any user-defined events that relate to database events follow the same naming convention for the event name.

Another way is to check the contents of the `_event-detail` field for a specific format, as shown:

```
DEFINE VARIABLE lDbEvent AS LOGICAL NO-UNDO.
IF _event-detail <> "":U AND
  NUM-ENTRIES(ENTRY(1,_event-detail,CHR(7)),CHR(6)) = 4 THEN
  lDbEvent = TRUE.
```

A similar approach could be used to identify other types of events with a specifically formatted `_event-detail`. The primary consideration is to ensure the event definition follows some sort of naming convention to make all this possible and to be able to fall back to the event definition itself to determine the format of audit-related fields.

Internal audit event policies

A number of internal audit events are defined to provide auditing for things like schema changes, audit policy changes, security changes, and database administration functions, for example.

In many cases the information about the internal events needing to be audited is represented as standard metaschema in an OpenEdge database in tables that all begin with underscore and have a negative table number. For example, changing schema results in modifying records in the `_file`, `_field`, `_index`, `_index-field`, `_file-trg`, `_field-trig`, `_sequence`, `_db`, `_db-detail`, `_db-option`, etc. tables, and changing audit policy involves updating records in the `_aud-*-policy` tables, and so on.

The audit policy tables are well-defined and facilitate fine-grained control over application tables and fields (`_aud-file-policy` and `_aud-field-policy`). These tables drive internal audit event behavior by simply defining audit policies for the metaschema tables. For example, creating a record in the `_aud-file-policy` table for the `_file` metaschema table itself controls auditing of table creates, deletes, and updates. Creating records for the `_aud-*-policy` tables drives auditing of audit policy changes.

OpenEdge includes standard audit policies that define the policy required to drive these internal audit events. OpenEdge Release 10.1 also includes standard reports to assist you with querying the audit data for these internal events. These predefined reports will relieve you from needing too much knowledge of the metaschema tables themselves.

For a list of supported internal audit events, start Audit Policy Maintenance and view the preconfigured OpenEdge audit policies, see the Audit Policy Maintenance Help, or seePreconfigured Audit Policies on page 235.

Efficient index use

Since the amount of audit data recorded can be extremely large, it is important to make effective use of the available indexes to efficiently query and report on the data. Note that the indexes can be disabled on the operational database for performance reasons.

The indexed fields on the _aud-audit-data main table are as shown in the following table (assuming indexes are enabled).

Table 9: _aud-audit-data main table indexed fields

Index name	Flags	Field	Description
_Data-guid	primary unique	_Audit-data-guid	Primary unique index. Joins the child table _Aud-daudit-data- value that holds the modified fields old and new values. This index is always active.
_Connection-id	–	_Database-connection-id _Client-session-uuid	Queries audit data by database connection, and further within a database connection by client session.
_Userid	–	_User-id	Queries audit data by the real application user who created the audit data.
_Event-group	–	_Audit-event-group _Db-guid _Transaction-id _Transaction-sequence	Queries data by some logical group (for example, business entity, task, workflow) and then within that to sequence the data by database transaction ID and sequence (that is, the order in which the data was created within the group or database transaction). Note that the use of groups is optional.
_EventId	–	_Event-id	Queries audit data by audit event ID; event IDs between the range 5000 to 5099 represent schema change events.
_Audit-time	–	_Audit-date-time	Queries audit data by date and time it was created (plus time zone). This index is always active.

Index name	Flags	Field	Description
_Event-context	–	_Event-context	Locates audit data for a specific record or thing. For database events, this would contain a delimited list of identifying field values to identify the originating record, such as a customer number.
_AppContext-Id	–	_Application-context-id	Queries audit data that occurred within the bounds of a particular application context, such as what audit data was generated by an object (procedure).

The indexed fields on the _aud-audit-data-value child table are listed in the following table:

Table 10: _aud-audit-data value child table indexed fields

Index name	Flags	Fields	Description
_Continuation-seq	Pu	_Audit-data-guid _Field-name _Continuation-sequence	The primary unique index for the table; used when joining from the parent _Aud-audit-data table to determine the modified fields for a particular audit data entry.
_Field-name	–	_Field-name	Facilitates querying audit data for changes to a specific field.

Note that the indexes can be deactivated for performance in the short-term operational (application) database where audit events are being captured. Note also that not all of the indexes are deactivated: the _Data-guid and the _Audit-time indexes are always active, as these are required by the utilities in all cases.

Query guidelines

The best ways to query the audit data depend on the selection criteria being used and the order that the data needs to be reported in.

Ordinarily, the primary query table will be the _aud-audit-data table, as this contains the most useful indexes. If searching for changes to a particular field, however, you will find it more efficient to start with the _aud-audit-data-value table and to join to the _aud-audit-data table using the _Audit-data-guid. Note that separate child records in the _aud-audit-data table are not always available, depending on the field and field policy settings.

The audit data table itself has two recursive joins, one for the application context and one for the audit event group, pointing at another audit data record containing the details and joined using the _Audit-data-guid primary key. Recursive joins are currently not natively handled in ProDataSets, so manual coding is required to handle this.

One approach is to add code to a ProDataSet `afterrowfill` event procedure to manually populate additional temp-tablefields with the context and event group information inline, using buffers to read the information. A variation of this approach is to add separate temp-tables for the context and audit event group records, which might be useful if all the data in the record is required for reporting. The important thing is to take away the complexity of the recursive joins for the consumer of the report data.

It is likely that the same context or audit event group will span many audit data records, so rereading this data each time could be inefficient. Other possibilities are to make a separate pass through the audit data for the context and audit event groups specifically, or to build up a list of the audit event group and context IDs used across the extracted audit data and to make a second pass using this list as the driver to obtain the additional information required. The best approach will depend on criteria for report performance and how the context and audit event groups have been used.

Give similar considerations to the client session records, as there will be many audit data records for a single client session.

Reporting on audit data with prefiltered and custom reports

As audit data accumulates in the auditing tables, you can generate audit reports about the data based on criteria you specify. There are two basic types of reports:

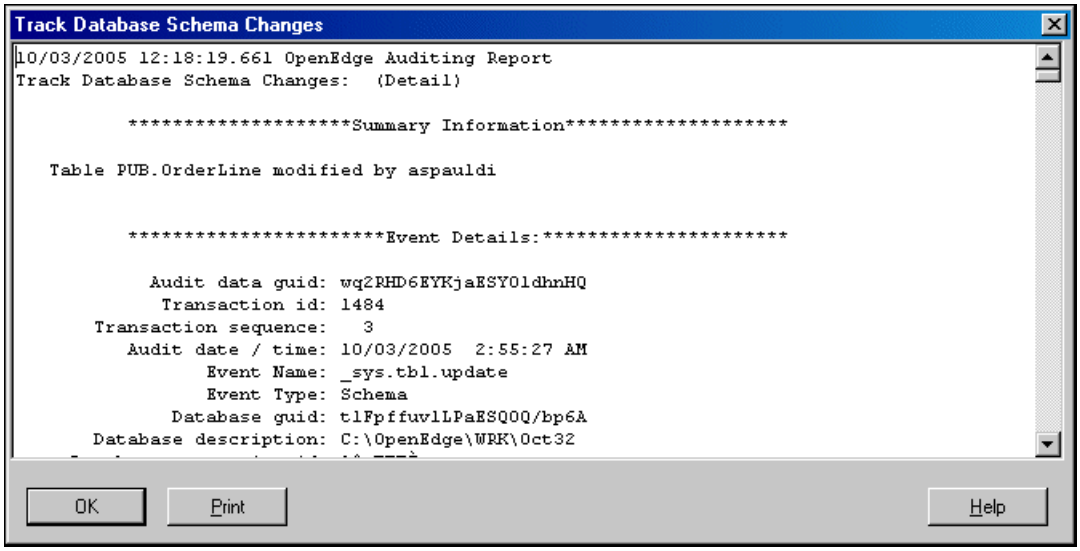
- **Prefiltered reports** — Default reports organized by category. You can specify the date range you are interested in capturing in each report. (A list of the prefiltered reports appears at the end of this section.)
- **Custom filtered reports** — Reports whose filter criteria, including date range, you establish.

To run these reports, you must have imported the preconfigured OpenEdge policies. For more information, see the Audit Policy Maintenance Help.

You can run the reports on all supported platforms. From Data Administration or the Data Dictionary, you can choose which report to run, select a specific date range, decide what kind of output you want (terminal, printer, XML file, or text file), and choose how you want the report contents to display (in detail or summary format). You can assign each report a name, and you can decide if you want to append a new report to an existing one. You can also determine whether the reports page length is fixed or continuous.

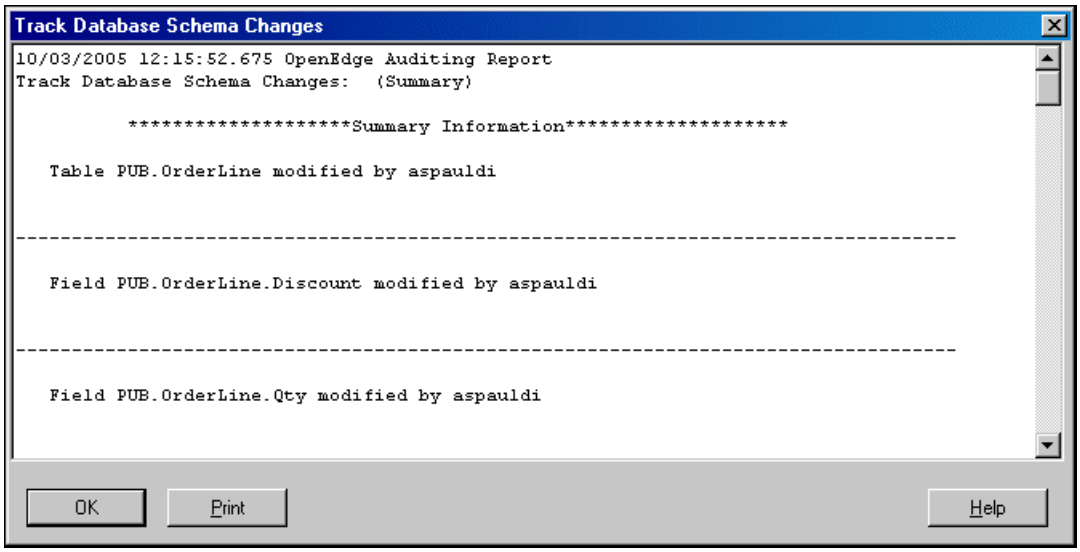
The following figure shows a report in detail orientation.

Figure 8: Detail orientation report output



The following figure shows a report in summary orientation.

Figure 9: Summary orientation report output



To choose the specific report you want to run, choose **Database** > **Reports** > **Auditing Reports**; then either select from one of the prefiltered report types listed or choose to generate your own custom audit data filter report. For the specific steps you follow, see the Data Administration Help.

The following table lists each of the prefiltered reports and describes the information the report contains.

Table 11: Prefiltered audit data reports

Report name	Description	Events
Track Audit Policy Changes	Reports on audit policy changes	• Audit Policy Maintenance Events: record created, updated, or deleted • Audit Policy Commit Events: audit policy run-time update/ commit changes • Audit Policy Dump or Load Events: dump and load of audit policies

Report name	Description	Events
Track Database Schema Changes	Reports on all possible schema modification actions	• Table created, updated, or deleted • Table trigger created, updated, or deleted • Table field created, updated, or deleted • Table field trigger created, updated, or deleted • Table index created, updated, or deleted • Table index field created, updated, or deleted • Sequence created, updated, or deleted • Database property created or updated
Track Audit Data Administration (Dump/Load)	Reports on all archiving and loading of audit data	Audit archive or audit data load
Track Application Data Administration (Dump/Load)	Reports on archiving and loading of audit data	Database text archive or load
Track User Account Changes	Reports on user account activity	User account created, updated, or deleted
Track Security Permission Changes	Reports on privilege definition and privilege assignment activity	• Privilege definition created, updated, or deleted • Privilege assignment created, updated, or deleted
Track SQL Permissions Changes	Reports on all possible SQL privilege modifications	• SQL database administrator created, updated, or deleted • SQL table privilege created, updated, or deleted • SQL column privilege created, updated, or deleted • SQL sequence privilege created, updated, or deleted

Report name	Description	Events
Track Authentication System Changes	Reports on authentication system maintenance and authentication system domain maintenance activity	• Authentication System Maintenance Events: authentication system created, updated, or deleted • Authentication System Domain Maintenance Events: authentication domain created, updated, or deleted
Client Session Authentication Report	Reports on all application access through authentication domains	Each time an OpenEdge application uses an authentication system to authenticate a user
Database Administration Reports (Utilities)	Reports on activities performed through command-line database administration utilities	• Database started, stopped, backed up, restored, or copied • Database binary dump or binary load • Database table moved • Database index moved, index checked, index rebuilt, or index fixed • Database area truncated • Database SQL dumped or loaded
Database Access Report (Login/Logout/etc.)	Reports on all login- and logout-type activities on an OpenEdge database	• Database user identity set or failed • User login success, user logout, or user login failed • SQL user login success, logout, or user login failed • Database connect or disconnect • SQL database connect or disconnect

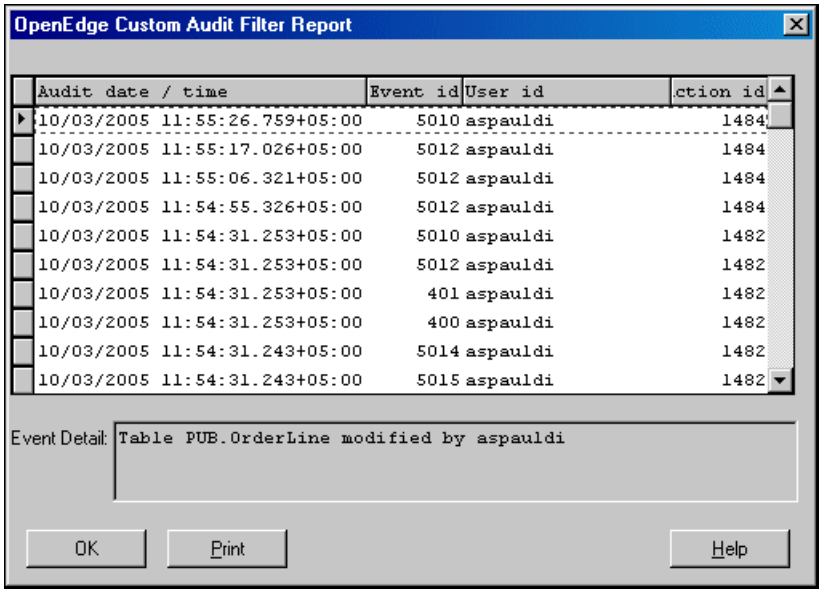
Generating a custom filtered report

You can generate a report in which you filter the recorded data by criteria you select. For example, you can generate a report based on an audit data range, an audit event ID range, event-context matching, or by user ID or transaction ID. You can output the contents to the screen in a tabular format or dump the output to XML or text format.

If you choose to generate a custom report and you select no filter criteria, you generate a report of all auditing activity. In this case, a warning appears before the report is created to remind you that generating a report based on all the auditing activity can take a long time and create a very large file. You can choose to generate the full report, or you can choose to go back and supply filter criteria.

The following figure shows a sample custom audit data filter report (in a Windows platform) in which Terminal was chosen as the output and Detail was chosen as the orientation.

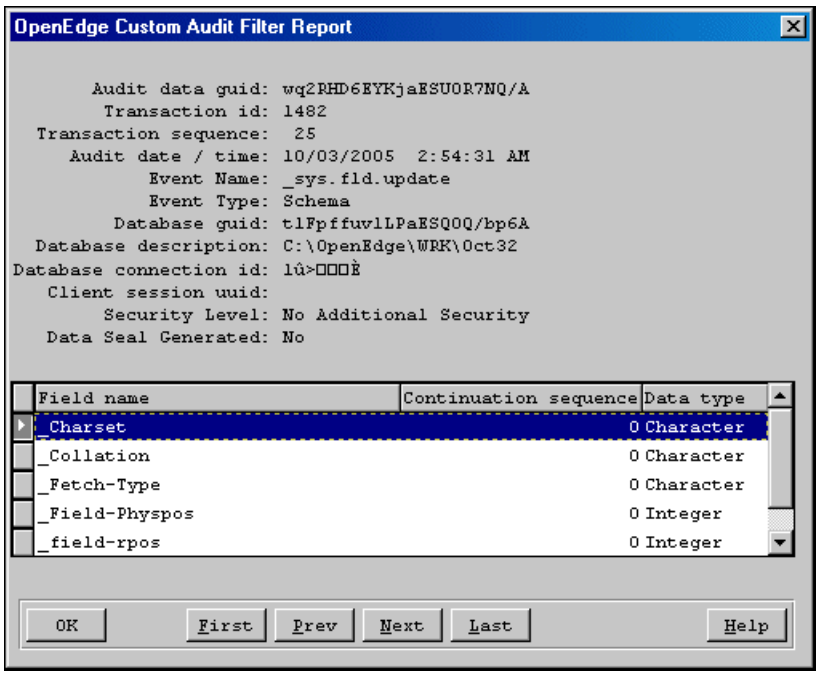
Figure 10: Sample custom audit data filter report



By selecting a column heading, you can view the report sorted in different orders (by date or time, for example). On UNIX, you can also sort the report by choosing the order: by Event Date/Time, Event Id, User Id, or Transaction Id.

You can view details about a particular record by selecting the record. The record details appear as shown in the following figure:

Figure 11: Custom audit data filter report detail



You can review the details about each record and sort through the records in the order you want.

IV

Transparent Data Encryption

For details, see the following topics:

- Transparent Data Encryption in OpenEdge RDBMS
- OpenEdge Key Store
- Configuring Transparent Data Encryption policies

14

Transparent Data Encryption in OpenEdge RDBMS

As part of an overall security strategy, Transparent Data Encryption provides for data privacy while the data is "at rest" in your OpenEdge database, regardless of the location of the database and who has a copy of it. The Transparent Data Encryption product must be installed in conjunction with an Enterprise Database license.

For details, see the following topics:

- Defining the problem space
- Defining Transparent Data Encryption
- Data security
- Before you start
- What is encryptable
- Transparent Data Encryption feature summary
- Enabling Encryption

Defining the problem space

When data is lost or stolen, there can be serious consequences. Data can be lost in many ways from the simple to the complex. For example, a laptop with sensitive data can be stolen, or hackers can breach corporate security through a deliberate attack. In the end, lost data can result in a loss of revenue, a loss of customer confidence, or even financial and legal penalties.

An increasing number of government regulations, industry standards, and company liability issues are driving corporations to require that their software vendors support data privacy "while at rest" and "when in transit". Database encryption is just one aspect of the larger requirement to provide end-to-end application data privacy, fulfilling the "while at rest" need.

To satisfy end-to-end data privacy requirements, database and middleware vendors apply data encryption, user authentication, and user authorization technologies at key points throughout their architecture. A clear text copy of the data can only be provided to an authenticated client who has explicitly been granted access to that data. Transparent Data Encryption enables OpenEdge application partners and direct end-users to complete end-to-end data privacy requirements.

Providing data privacy

Transparent database encryption is an essential part of providing end-to-end data privacy, enhancing the existing OpenEdge security features. However, data privacy is not an on/off solution. Each OpenEdge application must utilize and integrate the available security features to realize the optimal balance of security, performance, and administrative complexity for their situation.

Transparent Data Encryption operates with data-at-rest, using standard encryption libraries and encryption key management to provide transparent encryption of information in the database. Transparent Data Encryption protects data at the object level (table, index, LOB, type I area), so that one, some objects can be encrypted without encrypting the entire database. Protection is configured by defining policies, much as auditing levels are configured.

Using Transparent Data Encryption, data or index blocks written to disk are encrypted for safe storage, and data or index blocks read from disk are decrypted for use in memory. Data written to backups are encrypted, and binary dumps can be encrypted. This means that copies of the database, or encrypted backups or dumps are also protected.

Encryption key management is critical to an encrypted databases security. OpenEdge Transparent Data Encryption includes both policy tools and a secure encryption key store. Encryption key storage is kept separate from the database so that a stolen copy of the database does not contain the data's encryption keys. The key store is encrypted and protected by requiring account access with a strong passphrase to prevent unauthorized access and limit authorized access, further ensuring the safety of your data. See OpenEdge Key Store on page 177 for details on the OpenEdge key store.

Defining Transparent Data Encryption

Controlling access to private data while "at rest" (that is, stored in a file on disk inside your database), is the core of OpenEdge Transparent Data Encryption. OpenEdge combines various cryptography technologies and processes to give a security or database administrator control over who can access the private data within the database. OpenEdge Transparent Data Encryption is a smart choice to satisfy your need to secure data at rest because it offers a relatively low cost solution that is easy to implement.

Transparent Data Encryption provides three important elements:

- **Transparency** — Transparent data encryption means all data encryption is performed at run-time by the OpenEdge RDBMS, without any physical changes to ABL or SQL application code or database design. Application code executes without being aware of whether the database is or is not encrypting its data.
- **Configurational** — Configurable data encryption allows you to balance the database encryption and administration workload against individual security requirements. Transparent Data Encryption allows the DBA to configure encryption for just those database objects that require it. In this way OpenEdge can protect sensitive data at the lowest possible cost.
- **Security** — Secure data encryption ensures that once the OpenEdge RDBMS encrypts the data, it remains inaccessible to all except to those what have legitimate access to it.

The security environments that a transparent data encryption database must run in can vary. Therefore, transparent data encryption is configurable, allowing you to balance your databases encryption performance and administration workload against your security requirements. You can configure encryption for just the database objects that require it and adjust the level of encryption security to only what is necessary to be compliant with regulations.

Care has been taken in the design of OpenEdge Transparent Data Encryption to allow you to add encryption to your database without the need for you to become an expert in designing or implementing encryption technologies. The encryption is transparent to the user and it does not require any changes to existing ABL or SQL applications in order access encrypted data. Using it can be as simple as enabling and configuring the feature, migrating your unencrypted data, and then resuming your normal production operations.

Transparent data encryption utilizes symmetric key encryption to encrypt private data at the block level. Encryption of your database objects is managed through encryption policies. When you create an encryption policy, you can configure the cipher algorithm and encryption key length for each encrypted object. The creation and maintenance of encryption policies is discussed in Configuring Transparent Data Encryption policies on page 185.

Transparent data encryption depends on secure storage of encryption keys, and user access controls to your encryption keys to ensure that your datas encryption cannot be reversed by anyone other than those granted access. Each encrypted database has a single, unique Database Master Key (DMK). The DMK is created and managed by your database administrator and stored in your database key store, which is separate from your database. Your *key store* is an independent and secure entity that provides secure storage of data encryption keys and controls access in the form of user accounts. Details of the OpenEdge key store are discussed in OpenEdge Key Store on page 177.

Data security

Data security as pertains to Transparent Data Encryption applies in many areas, as discussed in the following sections:

- Data Storage on page 167
- Encrypted Data Configuration on page 168
- Encryption Key on page 168

Data Storage

Transparent data encryption ensures that your data's encryption cannot be reversed by anyone except for trusted database user accounts and database administrators. Transparent Data Encryption encrypts a databases private data to ensure that the private data remains encrypted in all of its storage locations.

The granularity of encryption is at the database storage object level. You can selectively configure individual database storage objects for encryption. Database objects where private data can be encrypted are:

- Type II storage area objects—tables, LOBs, indexes—at the object level.
- Type I storage area—designated at the level of granularity of the area. This means that **all** tables, LOBs, and indexes in an encrypted Type I storage area are encrypted.
- AI notes files
- BI notes files

A databases private data can be exposed to intruders outside the physical database storage. To retain data privacy (encryption) OpenEdge also enables data encryption for:

- PROBKUP file sets
- OpenEdge audit data
- Intermediate data transfer files used for binary dump and load and Audit archive and load. Note that for these transfer files, container security is utilized. This is different than object security and uses a different cipher than the object(s) in the transfer file.

Certain database objects cannot be encrypted. These restrictions are:

- Any (OpenEdge or application) table or index in the Schema area (Area 6)
- Any tables and indexes in the area Encryption area, where encryption policies are stored. Users are prevented from defining objects in the Encryption area.
- The Control area (Area 1)
- TL areas (for 2-phase commit)

For ABL and SQL clients, temporary files can contain unencrypted versions of encrypted data. Temporary files are always removed when the client exits. The `-t` startup parameter to save temporary files is not allowed for encryption-enabled database connections. ABL clients will not start if `-t` is specified; SQL clients ignore the parameter.

Note: Access to data (decrypted or never encrypted) can be restricted through the use of run time table and field permissions. For more information, see *OpenEdge Deployment: Managing ABL Applications* or *OpenEdge Data Management: SQL Development*.

Encrypted Data Configuration

Transparent Data Encryption stores its configuration in a set of security policy table records. These security policy records must be securely stored and administered. The security of policies is guaranteed by:

- Storing security policies in a separate Type II storage area that has special built-in ABL, SQL, and database utility access controls
- Disallowing direct record access by either ABL or SQL language clients to security policy table records
- Allowing security policy table records to be administered only by an authenticated user via:
  - SQL DDL language statements (SQL database administrator)
  - ABL [system] object methods from within the Data Dictionary or Data Administration, connected as a single user or shared memory connection to the database (Security administrator)
  - PROUTIL commands executed on the system where the database is located (ABL or SQL database administrator)
- Specifying OpenEdge auditing events and reports to track Security policy administration

Encryption Key

The security provided by physically encrypting data is only as strong as the security measures taken in the generation, storage, and access to the encryption keys. Transparent Data Encryption ensures strong data encryption by:

- Using a single database master encryption key that is stored externally from the database in the OpenEdge key store (*dbname*`.ks`) file.

- Not including the database master key and object encryption keys in any backup or binary/data dump media. They are not copied with the PROCOPY command either.
- Enforcing strong passphrase protection on the OpenEdge encryption key store. The passphrase must be entered whenever starting an OpenEdge database server or connecting to an encryption-enabled database single user.
- Not allowing anyone, including database administrators, direct access to encryption key values.
- Applying multiple levels of encryption to the stored encryption keys.
- Guaranteeing that database object encryption key values are unique per database and per database object in order to limit the window of data exposure in the event that an encryption key is obtained by an intruder.
- Prohibiting encryption keys from being exported, imported, or shared between multiple databases. An exception to this rule is between the database and its Replication and hot standby databases, which must share the same key store.
- Preventing database encryption key values from being transported across any OpenEdge network connection, regardless of whether SSL/TLS is utilized.
- Disallowing remote ABL database clients to administer database encryption keys, via their database object security policy records, where this sensitive data may be compromised (including Data Dictionary clients).
- Providing OpenEdge auditing events configurable to track access and administration of encryption keys and their storage.
- Implementing strong passphrases rules for OpenEdge key store.

For more information on the OpenEdge key store, see OpenEdge Key Store on page 177.

Before you start

Before you get started with Transparent Data Encryption, you should understand the following:

- **Know what objects in your database need to be encrypted**

  OpenEdge Transparent Data Encryption gives you the flexibility to select which objects in your database need to be encrypted. You should select the smallest set of objects that contain private data. Knowledge of your database schema is required to select the appropriate objects. You will also need to consider the indexes of the encrypted objects, based on the fields that comprise the index. If your index contains critical (private) fields of an encrypted table, you should encrypt the index.

- **Decide your AI and BI encryption**

  When you enable transparent data encryption, by default your BI files and AI files (if enabled) are also enabled for encryption. Progress Software Corporation strongly encourages you to encrypt your BI and AI files. Failure to encrypt your BI and AI files exposes your encrypted data in an unencrypted form in your BI and AI notes. If you decide to risk your AI and BI files, you can disable AI and BI encryption.

- **Choose the cipher(s) that meet your requirements**

  OpenEdge Transparent Data Encryption supports six different ciphers which vary in strength and performance. You will need to understand your requirements to pick the correct cipher; the stronger the cipher, the harder to break, but it also takes longer to encrypt and decrypt your data. For a general discussion of ciphers, see Cryptography on page 59. For a list of the object ciphers supported for transparent data encryption, see Creating encryption policies with PROUTIL EPOLICY on page 186..

- **Determine access to the database key store**

To open an encryption-enabled database, a user must be authenticated as able to open the database key store. The key store is created when you enable your database for encryption. See OpenEdge Key Store on page 177 for a detailed discussion of the OpenEdge key store. There are two ways to authenticate to the key store: manual start and autostart. With manual start, the user must supply a passphrase every time the database is opened.

For servers and utilities, an additional parameter (`-Passphrase`) is added to the command line to indicate that the user is to be prompted for a passphrase to open the key store. For ABL clients, the passphrase must be included in the CONNECT statement with the `-KeyStorePassPhrase` parameter. If the passphrase is authenticated, access is granted.

By configuring autostart, you are granting access to the key store without prompting for a passphrase to any user who can run connect to a database (single or multi-user) or run a database utility. You can override the autostart authentication by including the passphrase parameter. Manual start is more secure, but impacts automated database administration (scripts); autostart does not impact scripts, but gives unrestricted access to encrypted data.

Autostart has two levels of security, admin and user, that correspond to the two key store accounts. Admin access is required to add or update encryption policies or modify the key store. User access allows for the encryption and decryption of data with the existing encryption policies. When For more details on autostart, see Configuring Transparent Data Encryption policies on page 185.

Note: Databases started with the AdminServer or the operating system Cluster Resource Manager can not be configured for manual start. There is no valid way to prompt for the passphrase in these situations. You can add an encryption-enabled database, that has been started using a script, as a scripted database in OpenEdge Explorer or OpenEdge Management.

What is encryptable

OpenEdge allows encryption of the following database elements:

- Type I areas
- Type II table
- Type II index
- Type II LOB

Note: You cannot encrypt the Schema area of your database. You can encrypt your audit data tables.

OpenEdge encrypts the following storage files:

- BI files
- AI files
- Backup files
- Binary dump (if requested)
- Audit archive (if requested)

BI and AI files are encrypted by default. You can disable their encryption, but doing so is highly discouraged, because doing so creates the possibility that unencrypted copies of your encrypted data is stored in the BI and AI files. Backup files are always encrypted. Binary dump (and audit archive) files are not encrypted by default, even if the data being dumped is encrypted. You can encrypt your dump files with a password based encryption (PBE) cipher by adding an additional parameter to the command. If you intend to keep your dump or archive for an extended period of time, or transfer it, encrypting it may be prudent. For more information on encrypting a dump file, see *OpenEdge Data Management: Database Administration*.

Data considerations

Choosing the correct data to encrypt is specific to your database and application requirements, but consider the following when selecting objects to encrypt:

- For data in a Type I area, you must encrypt the entire area, even if only one field in one table must be encrypted. Consider moving the table with the critical field, and its indexes to a Type II area, so that you can encrypt only the critical table and indexes.
- For tables in a Type II area, understand the contents of the tables indexes to determine if the index needs to be encrypted. For example, if you have an employee table with a social security number field and a separate employee number field, if the social security number field is used in an index, the index should be encrypted. However, if you have an index on the employee number field, you may determine that it is not necessary to encrypt that index.

Transparent Data Encryption feature summary

To summarize, Transparent Data Encryption guarantees the following:

- Data in record or a database format (binary dump, backup) is encrypted. The output of an Export command, a Dictionary dump, or on the screen is not encrypted.
- The encryption cipher algorithms provided are industry standard.
- Configurable cipher specifications allow you to select the symmetric algorithm, mode, and key-size for each database object (table, LOB, index, area, AI, BI).
- Encrypted object encryption keys are derived from a single database master encryption key and a unique per-object value that yields a unique binary encryption key per object, per database.
- Access to database master encryption key and object encryption keys is restricted solely to the OpenEdge RDBMS storage engine. No direct user access is supplied, nor are these encryption keys ever transported over a network connection.
- The database encryption key is stored outside of the OpenEdge database and is protected by its own user accounts and access-control. Key store security regulates who has access to the database master encryption key and therefore the databases encrypted data.
- Transparent clear-text access to encrypted data is only available to an authenticated and authorized OpenEdge database user who also has the appropriate ABL or SQL run-time table and field access privileges, after a database server has been started by a database administrator with key store access.
- Replication and backup of encryption key store must done by the DBA using operating system tools.
- Online and offline configuration and maintenance of encrypted data is restricted to authenticated and authorized database administrators: a SQL DBA or ABL Security Administrator.
- Online and offline configuration and maintenance of OpenEdge key storage configuration is restricted to authenticated and authorized DBAs, with key store admin privileges, and may not be accessed over a network connection.

Enabling Encryption

When you are ready to enable your database for Transparent Data Encryption, there are two required steps, as discussed in the following sections:

- Adding an encryption area on page 172
- Executing the ENABLEENCRYPTION command on page 173

Note: For details on any aspects of your OpenEdge database or any commands to administer the database, see *OpenEdge Data Management: Database Administration*. If you are enabling encryption on a database that is also configured for OpenEdge Replication, see *OpenEdge Replication: User Guide* for additional instructions.

Adding an encryption area

Transparent Data Encryption requires a specific area to hold your encryption policies. To protect your policies, you cannot perform any record operations on the policy data with an ABL or SQL client. The encryption policy area has the following restrictions and characteristics:

- The area must be named "Encryption Policy Area" and the type token in your structure (`.st`) file must be an "e".
- The area number of the Encryption Policy Area must be greater than the area number of the Schema area.
- The area must be a Type II area.
- The first line defining the Encryption Policy Area in your structure (`.st`) file must contain both the area name and the area number.
- If the area definition in your structure file omits the area cluster size, the size defaults to the value of `CLUSTERSIZE_DEFAULT` (8 blocks per cluster).
- If the area number and the records per block values are omitted, the values are assigned following the rules for data areas.
- After the first definition line in the structure file, subsequent lines may omit the area name, area number, records per block, and cluster size values.

To add an Encryption Policy Area to your database:

1. Create a structure (`.st`) file describing the Encryption Policy Area. For example:

```
e "Encryption Policy Area":12,32;64 . f 1024
e "Encryption Policy Area":12,32;64 .
```

2. Add the Encryption Policy Area to your database with PROSTRCT ADD. For example:

```
prostrct add mydb encrypt_policy_area.st
```

3. Create a new structure file for your database that reflects the added Encryption Policy area with PROSTRCT LIST. For example:

```
prostrct list mydb
```

Executing the ENABLEENCRYPTION command

One command enables your database for transparent data encryption. Enabling encryption requires database administrator privilege. The basic syntax for enabling encryption is:

```
proutil dbname -C enableencryption [-Cipher cipher-num] [-Autostart { user | admin } ]
```

Enabling encryption performs many tasks on your database. When you enable encryption:

- The database BI is truncated if the database is offline and the BI is not already truncated.
- The schema for encryption policy area is loaded.
- New audit events for encryption are loaded.
- The OpenEdge key store is created, and the key store creates and stores the database master key. The key store is named, *dbname*`.ks`, and is stored in the same directory as your *dbname*`.db` file.
- The master database security record is created in the encryption policies.
- A UUID for the database is set, if it is not already.
- Encryption keys are generated for encrypting the database AI and BI files (unless explicitly turned off).
- Autostart is configured for the key store, if requested.
  - If you specify `user`, the key store user account is used for autostart.
  - If you specify `admin`, the key store admin account is used for autostart.
  - If you do not specify `-Autostart`, manual start is configured.
- You are prompted for passphrases:
  - The key store admin passphrase is required.

- The key store user passphrase is optional, but required if you specified `user` for the `-Autostart` parameter.
- The PBE passphrase is mandatory if you specify the PBE cipher for your key store (`-Cipher 6`).

By default, PROUTIL ENABLEENCRYPTION indicates that all future AI and BI notes are encrypted. If after-imaging is enabled, enabling encryption results in an extent switch. If you enable encryption while your database is online, BI notes are not encrypted until the next time the database is started. Existing AI and BI files are **not** encrypted; enabling encryption essentially sets an indicator for **future** writes.

By default, PROUTIL ENABLEENCRYPTION uses cipher 1, "AES_CBC_128". For details on ciphers, see Cryptography on page 59. The following table lists the supported database master key ciphers.

Table 12: Database Master key ciphers

ID	Cipher	Mode	Size	Key type
1	AES	CBC	128	binary
2	AES	CBC	192	binary
3	AES	CBC	256	binary
4	DES	CBC	56	binary
5	DES3	CBC	168	binary
6	DES	CBC	56	PBE

The key store passphrases, as well as the database master key passphrase if using the PBE cipher, require strong passphrase rules. Passphrases must conform to the constraints described in the following table:

Table 13: Passphrase constraints

Rule	Value
Minimum number of characters	8
Maximum number of characters	2048
Minimum number of numeric characters	1
Minimum number of alpha characters	2
Minimum number of punctuation characters	1
Character set	[a-zA-Z0-9]!@#$%^&*()_+-{}[]|\,./<>?;:<space>
First character	(see Character set)
Mixed case alpha required	True
Case sensitive	True

Once you have enabled your database for Transparent Data Encryption, you must define encryption policies before any data is encrypted. See Configuring Transparent Data Encryption policies on page 185.

15

OpenEdge Key Store

The OpenEdge key store provides secure storage of encryption keys in support of Transparent Data Encryption.

For details, see the following topics:

- General key store basics
- OpenEdge key store
- Configuring key store access
- Key store maintenance

General key store basics

The goal of Transparent Data Encryption is to encrypt selected database objects. To be successful, the encryption must meet certain industry security practice expectations with regards to encryption key storage, access-controls, and administration. This section introduces the general concepts of key store implementations.

The purpose of encrypting data is to bar its access to all but a few designated individuals. This requirement spans both run-time and at-rest storage. Encryption security (strength) is measured by how much time and effort is required for an unauthorized individual to recover the clear-text data. The encryption algorithm used has little effect on cryptography strength because they are all open-source; you can obtain the algorithm and code samples from many sources. Encryption security comes from:

- The size of the (binary) encryption key
- The randomness of the encryption keys value
- The user access controls that limit the encryption keys value to use only by designated individuals

In a practical sense, encryption is only as good as the security of the encryption key storage and its access controls. A key store implementation is successful when an intruder is unable to coerce it into yielding the stored encryption key values and therefore must resort to a brute-force attack on the encrypted data.

A secure key store is usually its own entity and provides its own user authentication and authorization. Most low-level key stores use simple passphrase access control; mid-level key stores enforce strong passphrase access controls; high-level key stores use stronger methods of access control such as LDAP, Kerberos, biometrics, or digital certificates.

There are no fixed rules or guidelines that influence the design of a key store and its physical storage. The majority of key store implementations are required to handle several hundred encryption keys and a small number user accounts. Portability is generally a key store priority as encrypted data is platform neutral and is simply copied from system to system. The key store must be capable of being transported from system to system, along with the encrypted data, using simple file copy operations. Most key stores rely on the underlying file system for robustness and disaster recovery.

Key stores fall into two general categories: single user and multi-user. Single user key stores store encryption keys for a single entity. A private key store usually has a single user account that provides all-or-nothing administration and encryption key access privileges. As a personal key store, it does not have to protect itself from multiple simultaneous processes or thread access. A multi-user key store has multiple user accounts and must survive in an environment where multiple processes and process-threads will compete for simultaneous access. Therefore, a multi-user key store has an administrator account for administration tasks and one or more user accounts for users who need access to encryption keys for their applications. The best practice for a key store is to be open as short a time as possible. The combination of short open times and very infrequent writes requires minimal file locking support, which is typically in the form of exclusive write access.

A key store always has the ability to store multiple encryption keys and provide a basic level of bookkeeping for them. All encryption keys eventually live past a useful lifetime, after which they are retired and replaced by a newer one. Typical bookkeeping includes items such as create and update history and active/retired status.

Key store access control is a high priority. First, encryption is used to protect the encryption keys and user accounts from intruders. Second, user access control grants access to an encryption key value only after a user has successfully entered their user-id and secret (most often a passphrase or equivalent). However, this can be in conflict with a business need to start an application (and access the encryption keys) without human intervention. Various means are used in the industry to provide an application "unattended startup" feature where the application can automatically access the key store and perform encryption. Many industry experts view "unattended startup" as insecure, but the practice remains as in many cases it is a business necessity.

A stored encryption key is simply an array of binary byte values. Encryption operations will not fail when you give it the wrong encryption key, you simply end up with random binary garbage. Therefore it is very important for the key store to provide the application with the ability to ensure that it is accessing the right encryption key. Most times this identification is provided by unique identifiers, but can also take the form of simple "alias" names.

Like any passphrase access controlled entity, a key store suffers from the classic "lost password" syndrome. When a key stores data is physically encrypted using a user account password, the key store is irrevocably lost when the password is lost (and subsequently the data is lost).

OpenEdge key store

Your OpenEdge database key store is created when you enable your database for transparent data encryption through the PROUTIL ENABLEENCRYPTION command. The key store has the following main functions:

- To store the Database Master Key (DMK) externally from the database.
- To derive the individual database object virtual keys from the DMK.
- To protect the DMK and object virtual keys from being copied.
- To control access to the key store through built-in user accounts with strong passphrase protection.

- To deny access to a transparent data encryption-enabled database if the user cannot open the key store by supplying a passphrase for one of the built-in key store user accounts.
- To configure opening of the key store through automated processes.

A key store has two built-in user accounts, the admin account and the user account. Administrator privilege is required to create or change any key store value, including all aspects of encryption key generation and storage, passphrases, and autostart configuration. User privilege is required to access encryption key values. You must always provide a passphrase for the admin account when you create the key store; the user account passphrase is optional. The passphrases for the user and admin accounts must be different.

For an introduction to PROUTIL ENABLEENCRYPTION, see Enabling Encryption on page 172. For complete details on the command, see *OpenEdge Data Management: Database Administration*.

When your key store is created, it is bound to your database but remains a separate entity. PROBKUP does **not** backup your key store. If you create a copy of your database with PROCOPY, the key store is not copied. The key store is not part of your database structure definition. If you copy an encryption-enabled database, you will not be able open the copy until you copy and rebind the key store to the copied database with the PROUTIL EPOLICY command.

Configuring key store access

To open a transparent database encryption-enabled database, OpenEdge must be able to open the key store. Successfully opening the key store requires that the correct key store passphrase is provided. If OpenEdge cannot open the key store, opening the database fails. You have two configuration choices regarding how the key store passphrase is supplied: manual mode or autostart mode. Manual mode requires that you supply (type in) a key store account passphrase any time your database is opened. Autostart delivers a passphrase you configure to open the key store automatically. Manual mode is more secure, but impacts automated database administration (scripts); Autostart mode does not impact scripts, but gives unfettered access to encrypted data.

Manual start

Once you have enabled encryption on your database, every time the database is opened the key store must be authenticated.

If you enable autostart, the authentication is done automatically. If autostart is not enabled, you must manually authenticate. To manually authenticate the key store, every utility, server, or client that opens the database must supply a passphrase. Manual start is best very secure installations where very few users need to open the database.

ABL clients specify a passphrase for key store authentication by adding the `-KeyStorePassPhrase` argument, followed by the passphrase to the CONNECT statement, as shown:

```
-KeyStorePassPhrase passphrase
```

It is the responsibility of the ABL application to handle prompting for the passphrase before executing the CONNECT statement.

Database utilities, servers, and single-user or shared memory clients indicate that they require a prompt for key store authentication by adding the following parameter to the command line:

```
-Passphrase
```

Note: If you have specified the key store user account for autostart, you can override your autostart authentication by adding `-Passphrase` or -KeystorePassPhrase, as appropriate for the command.

If access to the key store cannot be successfully authenticated, the database cannot be opened.

Autostart

Autostart delivers a passphrase you configure to open the key store automatically, so there is no prompting for a key store passphrase. Autostart mode does not impact scripts, but gives unfettered access to encrypted data.

Autostart can be configured when you enable your database for encryption by adding `-Autostart { user | admin }` to the PROUTIL ENABLEENCRYPTION command. If you do not specify `-Autostart`, manual start is configured. Specifying `-Autostart admin` is recommended only in a development environment. You can change the autostart configuration to use a different key store account with the PROUTIL EPOLICY MANAGE command at a later point in time.

To change your autostart configuration to the key store user account:

1. If you did not supply a key store user account passphrase when you enabled your database for encryption, create one now using PROUTIL EPOLICY MANAGE. You must specify the key store admin passphrase with `-Passphrase` if the database is not enabled for autostart with the admin account, as shown:

```
proenv>proutil t1demo -C epolicy manage keystore userphrase -Passphrase
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
Enter the key store passphrase for database t1demo:

This command modifies encryption access control in the Keystore file.
After successful completion of the command, the Keystore file must be
backed-up. (15518)
Enter new passphrase [required] :

Please Retype your Passphrase for Verification
Enter new passphrase [required] :

proenv>
```

2. Change your autostart configuration by entering the key store user account passphrase when prompted:

```
proenv>proutil t1demo -C epolicy manage autostart user -Passphrase
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
Enter the key store passphrase for database t1demo:

Manage autostart enable has been selected. (15523)
This command modifies encryption access control in the Keystore file. After
successful completion of the command, the Keystore file must be backed-up.
(15518)

proenv>
```

You can disable autostart at any time using PROUTIL EPOLICY MANAGE, as shown:

```
proenv>proutil t1demo -C epolicy manage autostart disable -Passphrase
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
Enter the key store passphrase for database t1demo:

Manage autostart disable has been selected. (15523)
This command modifies encryption access control in the Keystore file. After
successful completion of the command, the Keystore file must be backed-up.
(15518)
proenv>
```

Observe that `-Passphrase` is added to the command line to override the autostart user account configuration and use the admin account.

Key store maintenance

Proper care of your OpenEdge key store is vital to maintaining access to your encryption-enabled database.

Backups

The key store is not backed up as part of an OpenEdge PROBKUP. Therefore, it is crucial that the key store is backed up every time the database is backed up. However, the key store backup should not be co-located with the database backup. Informational messages also provide timely reminders to back up your key store when you perform regular maintenance.

Modifying passphrases

It is a good security practice to update passphrases on a regular basis. PROUTIL EPOLICY MANAGE KEYSTORE provides the ability to update the user and admin passphrases. The following example updates the user passphrase. You must specify the key store admin pass pr ha se with `-Passphrase` if the database is not enabled for autostart with the admin account, as shown:

```
proenv>proutil dest_db -C epolicy manage keystore userphrase -Passphrase
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
Enter the key store passphrase for database dest_db:

This command modifies encryption access control in the Key store file. After
successful completion of the command, the Keystore file must be backed-up. (15518)
Enter new passphrase [required] :

Please Retype your Passphrase for Verification
Enter new passphrase [required] :

proenv>
```

Observe that no message is provided to indicate that the passphrase was successfully changed. If the passphrase change is unsuccessful, error messages are provided.

The admin passphrase can be changed by substituting `adminphrase` for `userphrase`.

Rebind

Rebind allows you to bind an existing key store file to a new database GUUID created with the PROCOPY `-newinstance` command. The following example demonstrates PROUTIL EPOLICY MANAGE KEYSTORE REBIND. Observe that the first REBIND command fails because the key store is not copied with the PROCOPY command. There is no need to prompt for passphrases in this example because autostart is configured for the admin account:

```
proenv>procopy src_db dest_db -newinstance
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
Procopy session begin for shannon on CON:. (451)

Formatting extents:
   size                area name   path name
      8    Primary Recovery Area C:\OE102BWrk\dest_db.b1 00:00:00
     16              Schema Area C:\OE102BWrk\dest_db.d1 00:00:00
     16                Info Area C:\OE102BWrk\dest_db_7.d1 00:00:00
     16      Customer/Order Area C:\OE102BWrk\dest_db_8.d1 00:00:00
     16       Primary Index Area C:\OE102BWrk\dest_db_9.d1 00:00:00
     64      Customer Index Area C:\OE102BWrk\dest_db_10.d1 00:00:01
     64         Order Index Area C:\OE102BWrk\dest_db_11.d1 00:00:00
    256   Encryption Policy Area C:\OE102BWrk\dest_db_12.d1 00:00:00
     64   Encryption Policy Area C:\OE102BWrk\dest_db_12.d2 00:00:00
     16               Audit Area C:\OE102BWrk\dest_db_20.d1 00:00:00
     16             Audit Index C:\OE102BWrk\dest_db_22.d1 00:00:00

Copying src_db to dest_db... (6715)

Start writing data blocks. (6718)
01:15:53 10 Percent complete.
01:15:53 20 Percent complete.
01:15:54 30 Percent complete.
01:15:54 40 Percent complete.
01:15:54 50 Percent complete.
01:15:54 60 Percent complete.
01:15:54 70 Percent complete.
01:15:54 80 Percent complete.
01:15:54 90 Percent complete.
01:15:54 100 Percent complete.
1862 blocks copied. (6720)

...Copy complete. (6722)
Database copied from src_db. (1365)
Procopy session end. (334)

proenv>proutil dest_db -C epolicy manage keystore rebind
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
Key store validation error -449: (keystore) missing key-store file {02020364}
(15568)

proenv>cp src_db.ks dest_db.ks

proenv>proutil dest_db -C epolicy manage keystore rebind
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
Key store rebind: new database guid successfully created. (15512)
Key store rebind operations have completed successfully. (15514)
```

Note: KEYSTORE REBIND is not interruptible. If the operation does not successfully complete, you need to re-copy the original key store and re-issue the KEYSTORE REBIND command.

Key store reconstruct

If you created a key store using a password-based encryption cipher for the database master key, you can reconstruct the key store in the event that the key store is deleted or corrupted. Key stores created with any other master key cipher **cannot** be reconstructed. The following example shows the reconstruction of a key store with a PBE passphrase:

```
proenv>proutil pbe_db -C epolicy manage keystore reconstruct
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
This command modifies encryption access control in the Key store file. After s
cessful completion of the command, the Keystore file must be backed-up. (15518
Enter PBE passphrase [required] :

Please Retype your Passphrase for Verification
Enter PBE passphrase [required] :

Enter administrator passphrase [required] :

Please Retype your Passphrase for Verification
Enter administrator passphrase [required] :

Key store Reconstruct Completed. (15516)

proenv>
```

16

Configuring Transparent Data Encryption policies

Encryption of your database objects is managed through encryption policies. You define which objects are encrypted and the encryption cipher for the object. Policies are stored in your database in a designated Encryption Policy Area. Object policies utilize virtual data encryption keys derived from your DMK and the specified cipher. The encryption key for each encrypted database object is unique.

For details, see the following topics:

- Creating encryption policies
- Encrypting your existing data
- Maintaining transparent data encryption
- OpenEdge SQL support for transparent data encryption
- Disabling encryption

Creating encryption policies

Encryption of your data is managed through encryption policies. When you create a policy, you specify what database object (table, index, LOB, or Type I area) will be encrypted and the strength of the encryption cipher for the object. If you do not specify a cipher, the default, AES_CBC_128, is used. Creating a policy does **not** encrypt your existing data; it indicates that all future writes are encrypted. See Encrypting your existing data on page 189 for instructions on encrypting your existing data.

Encryption policies are created in several ways. See one of the following sections for more information:

- To create a policy from the command line, see Creating encryption policies with PROUTIL EPOLICY on page 186.
- To create a policy with Data Admin, see Creating encryption policies with the Data Administration tool on page 186.
- To create a policy with OpenEdge SQL, see OpenEdge SQL support for transparent data encryption on page 195.

Creating encryption policies with PROUTIL EPOLICY

The PROUTIL EPOLICY MANAGE command creates encryption policies for Type I areas and for objects in Type II areas. PROUTIL EPOLICY MANAGE requires database administrator and key store admin privileges. The basic syntax for creating an encryption policy is:

```
proutil db-name -C epolicy manage object-type encryptobject-name
```

OpenEdge supports the object ciphers described in the following table:

Table 14: Object ciphers

ID	Cipher	Mode	Size	Key type
0	NULL	NULL	—	—
1	AES	CBC	128	binary
2	AES	CBC	192	binary
3	AES	CBC	256	binary
4	DES	CBC	56	binary
5	DES3	CBC	168	binary

The *object-type* is one of the following: `area`, `index`, `lob`, or `table`, and the *object-name* is the name of the object, in quotes, if necessary.

For example, the following command creates an encryption policy with the default cipher on a Type I area named `DataArea100` in a database named `t1demo`:

```
proenv>proutil t1demo -C epolicy manage area encrypt "DataArea100"
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
Encryption policy setting for Area DataArea100 in Area 100
Cipher specification setting to AES_CBC_128 completed.
```

See *OpenEdge Data Management: Database Administration* for the complete syntax of EPOLICY MANAGE.

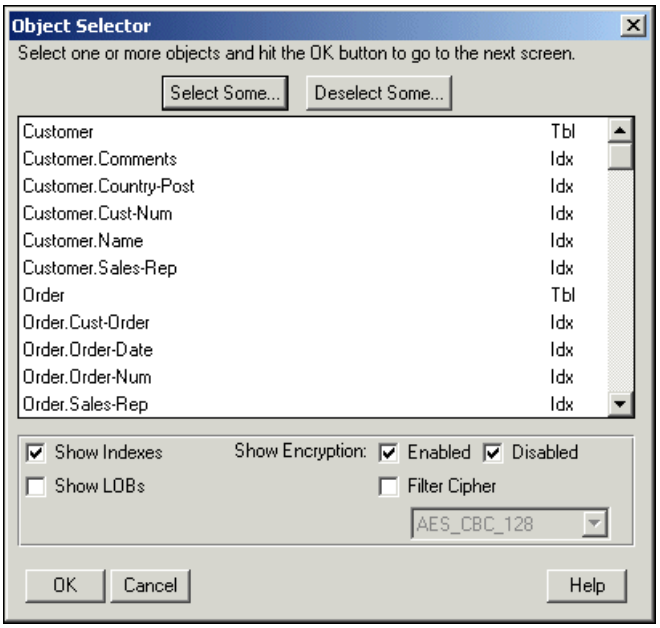
Creating encryption policies with the Data Administration tool

Create encryption policies for objects in your encryption-enabled database with the **Edit Encryption Policy** dialog. Objects must reside in a Type II area to be assigned a specific encryption policy. (You can encrypt an

entire Type I area with PROUTIL EPOLICY. See Creating encryption policies with PROUTIL EPOLICY on page 186 for details.) You must be connected to the database locally and have ABL security administrator and key store admin privileges to create encryption policies with Data Admin. The following steps present a guide for defining encryption policies.

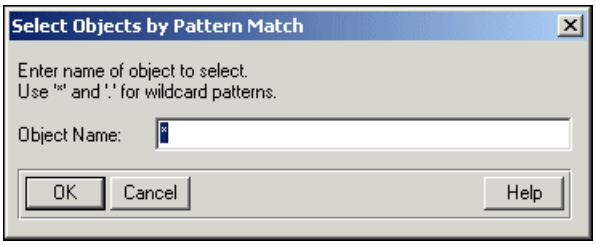
To define an encryption policy for a database object:

1. From the Data Administration tool, choose **Admin** > **Security** > **Encryption Policies** > **Edit Encryption Policy**. The **Object Selector** dialog appears:



   The initial view displays the tables and indexes of the connected database with encryption both enabled and disabled. You can change the list to view different sets of objects by checking and unchecking the **Show** boxes. For example, to also see LOBs, check **Show LOBs**; to only see objects with an existing encryption policy, for **Show Encryption** check **Enabled**, and uncheck **Disabled**.

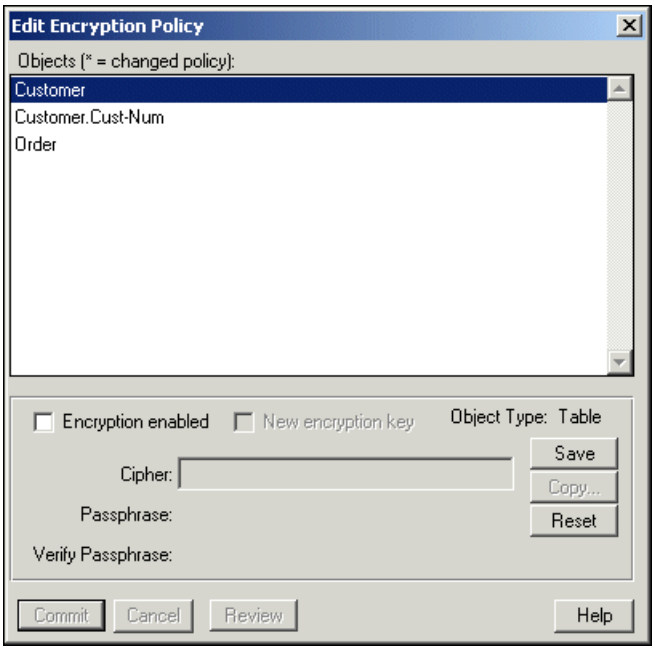
2. Scroll through the list, selecting objects, or click **Select Some** to bring up the **Select Objects by Pattern Match** dialog to refine the list:



   Enter the object name or pattern in the **Object Name** fill-in and click **OK**. When you return to the **Object Selector** dialog, any objects matching the pattern are added to the selected objects.

3. If you selected too many objects, clicking **Deselect Some** brings up the **Deselect Objects by Pattern Match** dialog, allowing you to enter an object name or pattern to deselect. Click **OK** to return to the **Object Selector** dialog with any objects matching the specified pattern deselected.

4. Once you have selected all the objects, click **OK** to proceed. The **Edit Encryption Policy** dialog appears:

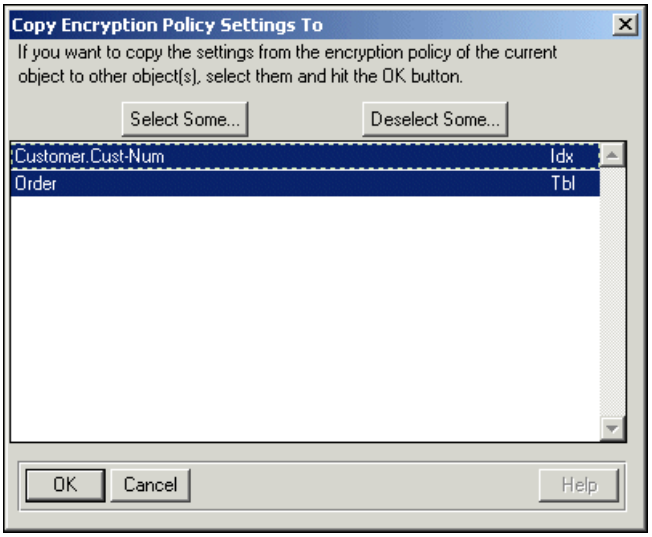


5. Select an object and check **Encryption enabled**. The default cipher, AES_CBC_128, is selected. Change the cipher if desired. Click **Save** to save the change, or **Reset** to undo the change.

Note: The change is not committed to the database until you click **Commit**.

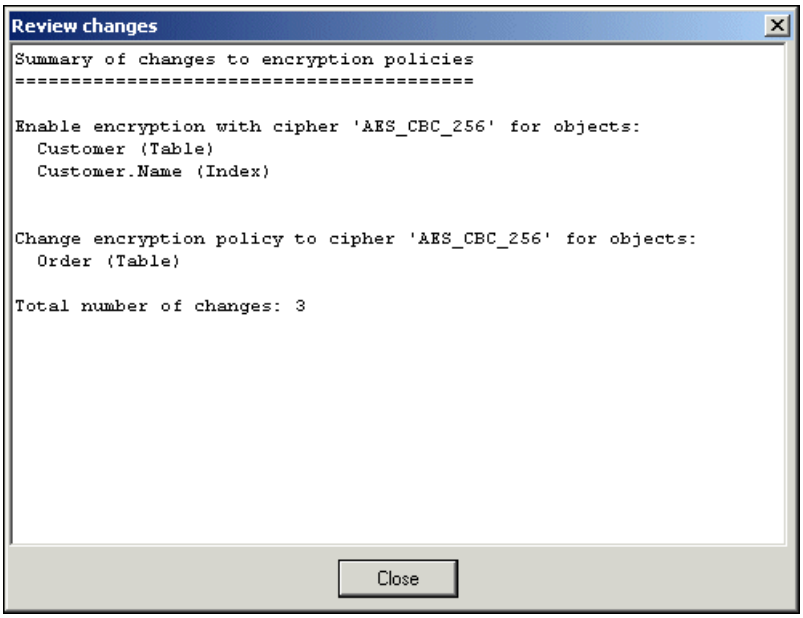
---

6. When you have saved a change, the **Copy** button is activated. Click **Copy** to propagate your change to multiple objects at once. Clicking **Copy** brings up the **Copy Encryption Policy Setting To** dialog, as shown:

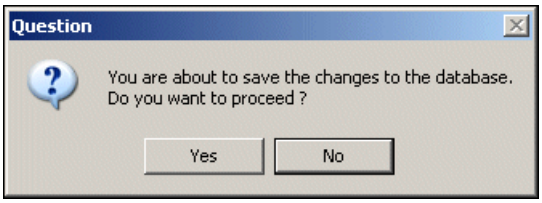


You can select a single object or multiple objects. When you have selected all you want to copy the setting to, click **OK** to return to the **Edit Encryption Policy** dialog.

7. Continue selecting and modifying object settings until you have specified all the objects you want to change. An asterisk (*) in front of the object name indicates you made changes to it. Click **Review** at any time to view the status of your changes, as shown:



8. When your changes are complete, click **Commit** to commit the changes to the database, or **Revert**, to cancel all your changes. If you choose to commit your changes, you are asked to confirm, as shown:



Encrypting your existing data

There are several different ways to initially encrypt or re-encrypt your existing data. The method you choose depends on the security needs of your database. Methods include:

- Allowing the normal course of database updates to encrypt the data. If a new encryption policy is applied, every time an block is updated, the block is encrypted with the new policy prior to being written back to the database. If the data you are encrypting is updated regularly, all the records are encrypted (for the first time, or with the new policy) during the normal course of operations.
- Dumping and loading your data. If you can interrupt your normal database operations, dumping and loading the database objects you have enabled for encryption encrypts all the data during the load.
- Encrypt the data with PROUTIL EPOLICY MANAGE UPDATE, as described in the next section.

EPOLICY MANAGE UPDATE example

In the following example, an encryption policy is set for the Type I area `DataArea101`. The normal course of updating records encrypts some of the blocks, and then EPOLICY MANAGE UPDATE completes the encryption. EPOLICY SCAN is used to verify that your data is becoming encrypted, and to check on the progress of the encryption. Execution of the EPOLICY commands requires database administrator and key store admin privileges. The general process is as follows:

1. Create an encryption policy for the area:

```
proenv>proutil t1demo -C epolicy manage area encrypt "DataArea101"
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
Encryption policy setting for Area DataArea101 in Area 101
Cipher specification setting to AES_CBC_128 completed.
```

2. Check the status of the area with EPOLICY SCAN:

```
proenv>proutil t1demo -C epolicy scan area "DataArea101"
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
AREA     DataArea101 / 101  CURRENT   AES_CBC_128  V:0   1 of 627 blocks encrypted
```

   Observe that at this point, only one block is encrypted.

3. Allow normal database processing to proceed and re-check the progress of encryption with EPOLICY SCAN:

```
proenv>proutil t1demo -C epolicy scan area "DataArea101"
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
AREA     DataArea101 / 101  CURRENT   AES_CBC_128  V:0   200 of 627 blocks encrypted
```

   At this point, 200 blocks have been encrypted.

4. Encrypt the remaining blocks in the area with EPOLICY MANAGE UPDATE:

```
proenv>proutil t1demo -C epolicy manage area update "DataArea101"
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
AREA     DataArea101 / 101  CURRENT   AES_CBC_128  V:0   427 of 627 blocks encrypted
```

   MANAGE AREA UPDATE reports encrypting the remaining 427 blocks of the area that were not previously encrypted.

5. Confirm that all the blocks in the area are encrypted with an additional run of EPOLICY SCAN:

```
proenv>proutil t1demo -C epolicy scan area "DataArea101"
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
AREA     DataArea101 / 101  CURRENT   AES_CBC_128  V:0   627 of 627 blocks encrypted
```

   All the data in the area is now encrypted.

Encryption of new data

If you set an encryption policy on an empty database object, as blocks are written to that object they are encrypted.

The following steps set an encryption policy on an empty Type I area, and then shows the status of the area after a table is defined and loaded with data:

1. Create an encryption policy for an empty area with EPOLICY MANAGE ENCRYPT:

```
proenv>proutil t1demo -C epolicy manage area encrypt "DataArea100"
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
Encryption policy setting for Area DataArea100 in Area 100
Cipher specification setting to AES_CBC_128 completed.
```

2. Create a table in the area and load data into it (not shown).
3. Check the encryption status of the area with EPOLICY SCAN:

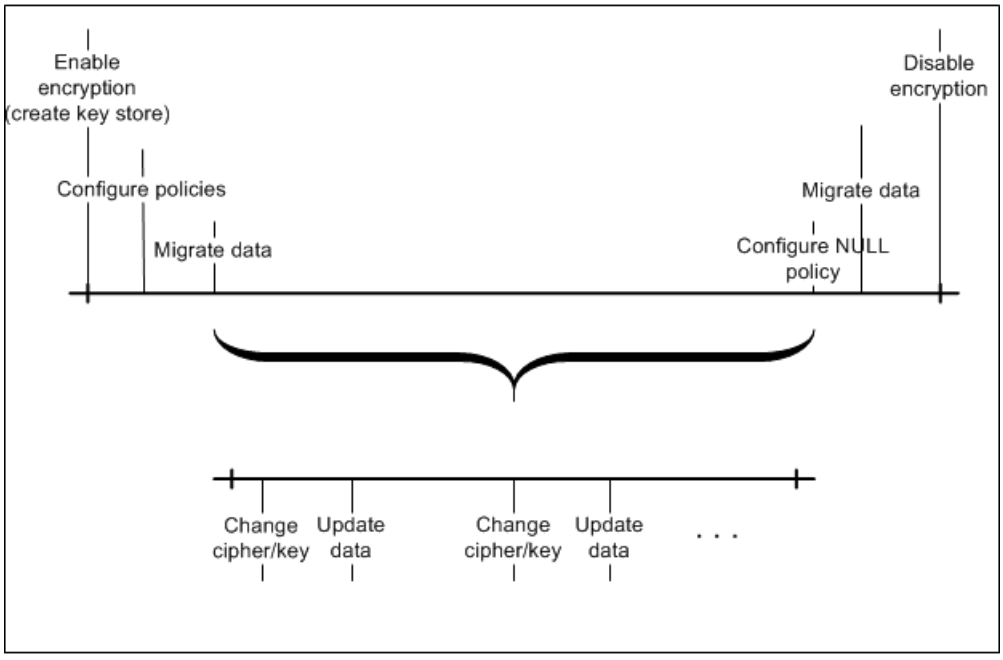
```
proenv>proutil t1demo -C epolicy scan area "DataArea100"
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
AREA     DataArea100 / 100  CURRENT   AES_CBC_128  V:0   127 of 127 blocks encrypted
```

As new records are written to the `DataArea100` area, they are encrypted. The output shows that the area has grown from 1 block to 127 blocks and that they are all encrypted.

Maintaining transparent data encryption

Once you have established your encryption policies and all your data is encrypted, you will need to perform some policy maintenance. Periodically, your encryption policies should be updated with a new key. A new key keeps the encryption cipher the same, but provides new input to the cipher algorithm. The following figure depicts the periodic updating of encryption policies in the life cycle of encrypted data.

Figure 12: Encrypted data life cycle



Encryption policies are rekeyed in several ways. See one of the following sections for more information:

- To rekey a policy with Data Admin, see Rekeying encryption policies with the Data Administration tool on page 192.

- To rekey a policy from the command line, see Rekeying an encryption policy with PROUTIL EPOLICY on page 193.
- To rekey a policy with OpenEdge SQL, see OpenEdge SQL support for transparent data encryption on page 195.

The following guidelines apply to object encryption policies:

- There can be no more than two active (current and previous) encryption policies associated with a database object.
- If one policy exists for a database object (the current policy), you can create a new version of the policy. The new version becomes the current policy and the other becomes the previous.
- If two policies (current and previous) exist for an object, you can not create another version of the policy until all data encrypted with the "previous" policy, is migrated to the "current" policy.

You can also view the history of an encryption policy. See Viewing encryption policy history in the Data Administration tool on page 194 for instructions.

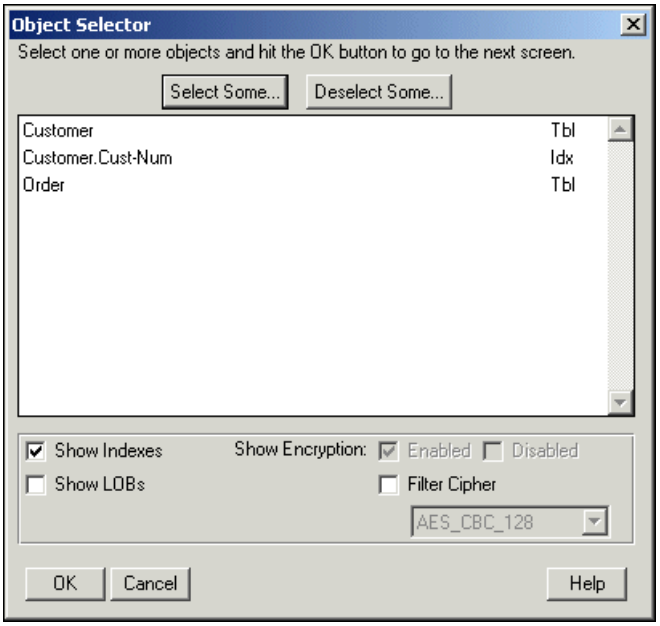
Rekeying encryption policies with the Data Administration tool

Generate new encryption keys for encryption policies in your encryption-enabled database with the **Generate Encryption Keys** dialog of the Data Administration tool. Objects must have an existing encryption policy to be rekeyed. You must be connected to the database locally and have ABL security administrator and key store admin privileges to rekey encryption policies.

Note: Generating new encryption keys is also supported in the character-mode Data Dictionary tool.

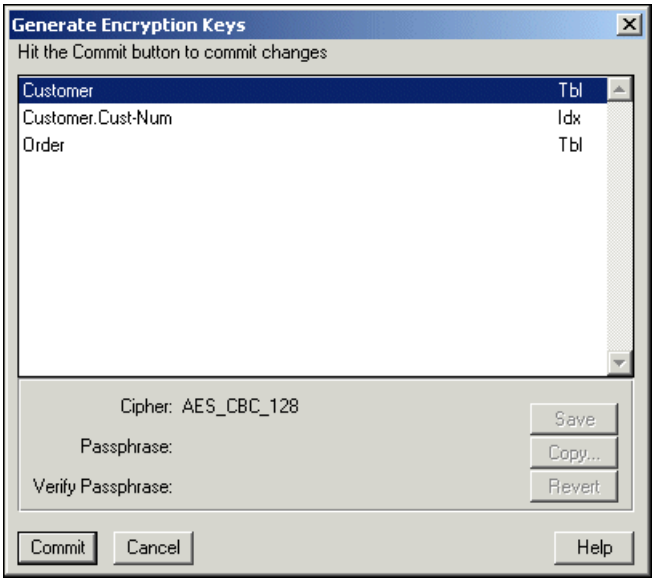
To generate a new encryption key for an existing encryption policy:

1. From the Data Administration tool, choose **Admin** > **Security** > **Encryption Policies** > **Generate Encryption Keys**. The **Object Selector** dialog appears:



By default, only tables and indexes with existing encryption policies are shown. You can also show LOBs with encryption policies by checking **Show LOBs**. You can refine the list by filtering by cipher if you check **Filter Cipher** and choose a cipher.

2. Select the object or objects you want to generate new encryption keys for and click **OK**. The **Generate Encryption Keys** dialog appears:



Note: The **Save**, **Copy**, and **Revert** buttons are disabled, as well as the **Passphrase** and **Verify Passphrase** fill-ins.

3. Click **Commit** to generate new keys for all the objects listed in the **Generate Encryption Keys** dialog, or click **Cancel** to not generate any keys.
4. If you click **Commit**, you are asked to confirm your decision.



Rekeying an encryption policy with PROUTIL EPOLICY

PROUTIL EPOLICY MANAGE allows you to rekey the cipher of an existing encryption policy. PROUTIL EPOLICY MANAGE requires database administrator and key store admin privileges.

To update the key of a policy, enter the following command:

```
proutil dbname -C epolicy manage object-type rekey object-name
```

You can only rekey the policy of an object that has previously been encrypted.

Changing the cipher of an encryption policy with PROUTIL EPOLICY

PROUTIL EPOLICY MANAGE allows you to change the cipher of a policy. PROUTIL EPOLICY MANAGE requires database administrator and key store admin privileges.

To change the cipher of a policy, enter the following command:

```
proutil dbname -C epolicy manage object-type cipher object-name-Cipher
cipher-num
```

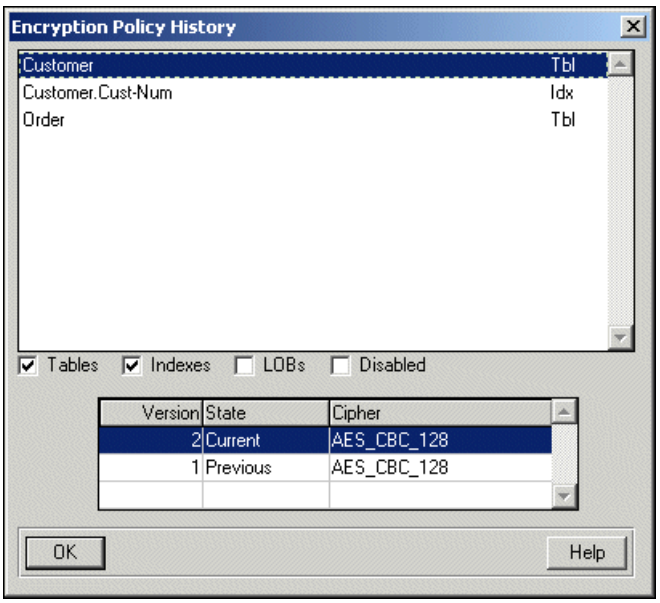
You can only rekey the policy of an object that has previously been encrypted.

Viewing encryption policy history in the Data Administration tool

You can view the history of an encryption policy in the Data Administration tool. Use the following steps to view encryption policy history for encrypted objects in an encryption enabled database.

To view encrytion policy history:

1. From the Data Administration tool, select **Admin** > **Security** > **Encryption Policies** > **Encryption Policy History**. The **Encryption Policy History** dialog appears.:



   The **Version** of a policy is an integer that starts at zero and increases every time a policy is modified. The **State** indicates whether the policy is the Current or Previous policy. The **Cipher** indicates the cipher of the policy.

2. In the **Encryption Policy History** dialog, select an object to view its encryption policy history. You can modify the listed objects by checking and unchecking the boxes for **Tables**, **Indexes**, **LOBs**, and **Disabled**.
3. When you finish reviewing encryption policy histories, click **OK** to exit the **Encryption Policy History** dialog.

OpenEdge SQL support for transparent data encryption

OpenEdge SQL Data Definition Language can be used to enable Transparent Data Encryption. This functionality provides **only** the DBA or security administrator, who also has key store admin privileges, with the ability to:

- Define encryption specifications for new tables, indexes, and LOB columns
- Modify encryption or decryption specifications for existing tables, indexes, and LOB columns
- Drop existing encrypted tables, indexes, and LOB columns
- View encryption specifications for tables, indexes, and LOB columns
- Rekey existing encrypted tables, indexes, and LOB columns

Note that the tables, indexes, and LOB columns must be in a Type II storage area.

Using the CREATE TABLE statement

Use the ENCRYPT WITH clause in the CREATE TABLE statement to create a new encrypted table. The syntax for the statement is:

```
CREATE TABLE [owner_name.] table_name
  ( {column_definition | table_constraint}, . . .)
    [ AREA area_name ]
    [ ENCRYPT WITH  cipher]
    [BUFFER_POOL { PRIMARY | ALTERNATE } ]
    [progress_table_attribute_keyword value ]
   ;

CREATE TABLE [ owner_name.] table_name
    [ (column_name [ NOT NULL] , . . . ) ]
         [ AREA area_name ]
         [ ENCRYPT WITH cipher ]
     BUFFER_POOL { PRIMARY | >ALTERNATE } ]
AS query_expression
     ;
```

The the ENCRYPT WITH clauses are highlighted in **bold**. For a list of supported values for *cipher*, see Table 1.

The following example illustrates the creation of a table using a the AES_CBC_192 encryption cipher:

```
CREATE TABLE tblname (fld1 int, fld2 int, fld3 varchar(25)) AREA areaname ENCRYPT WITH
 'AES_CBC_192';
```

When the ENCRYPT WITH clause is specified, OpenEdge SQL creates a new encryption policy for the table in the security schema. Data encryption does not occur until database object blocks are written to disk by an application transaction or by separate utilities. Note that the database must first be transparent data encryption-enabled to use the ENCRYPT WITH clause and that the new table must be associated in the statement with a Type II storage area.

Adding an encrypted column

You can create a new encrypted LOB column in a table by using either the CREATE TABLE or the ALTER TABLE ADD COLUMN statements with the optional ENCRYPT WITH clause. When the ENCRYPT WITH clause is specified, OpenEdge SQL creates a new encryption object policy for the new BLOB or CLOB column in the database security schema. The new column must be either a BLOB or a CLOB data type in a Type II area to be encrypted. Use the following syntax to create a new encrypted LOB column:

```
{ LVARCHAR | CLOB | LVARBINARY | BLOB } [ ( length ) ]
[ AREA areaname ]
[ ENCRYPT WITH cipher]
[BUFFER_POOL{ PRIMARY | ALTERNATE} ]
```

Using the CREATE INDEX statement

Use the CREATE TABLE statement with the optional ENCRYPT WITH clause to create an encrypted index. When using the ENCRYPT WITH clause, OpenEdge SQL creates a new object policy for the index in the database security schema. Encryption does not take place until database object blocks are written to disk in a later application transaction or through a separate utility. Note that the new index must be associated in the statement with a Type II storage area. To create an encrypted index, use the following syntax:

```
CREATE [ UNIQUE ] INDEX index_name
ON table_name
( { column_name [ ASC | DESC ] } [, ... ] )
[ AREA area_name ]
[ ENCRYPT WITH cipher]
[BUFFER_POOL{ PRIMARY | ALTERNATE} ]
[ PRO_DESCRIPTION value | PRO_ACTIVE {'N' | 'n'} ];
```

Using the ALTER TABLE statement

The ALTER TABLE statement can be used to change the security policies of an object. SET ENCRYPT, DECRYPT, and REKEY clauses can be used to define encryption policies for tables, indexes or LOB columns; the ALTER TABLE statement is an online operation. The ALTER TABLE statement has the following syntax:

```
ALTER TABLE [ owner_name. ] table_name
{ ADD column-definition
 | SET progress_table_attribute value
| SET [ ENCRYPT WITH cipher
| DECRYPT
| ENCRYPT REKEY ]
 | BUFFER_POOL { PRIMARY | ALTERNATE }
 | ALTER [ COLUMN ] column_name { SET DEFAULT value
                                | DROP DEFAULT
                                | SET [ NOT ] NULL
                                | SET progress_column_attribute value }
                                | SET ENCRYPT WITH cipher
                                | SET DECRYPT
                                | ENCRYPT REKEY
                                | SET BUFFER_POOL { PRIMARY | ALTERNATE } }
| DROP COLUMN column_name { CASCADE | RESTRICT }
| ADD CONSTRAINT constraint_name ] { primary_key_definition
                                   | foreign_key_definition
                                   | uniqueness_constraint
                                   | check_constraint } [ AREA area_name ]
| DROP CONSTRAINT constraint_name [ CASCADE | RESTRICT ]
| ALTER INDEX index_name { SET progress_index_attribute value

                         | SET ENCRYPT WITH cipher
                         | SET DECRYPT
                         | SET ENCRYPT REKEY
                         | SET BUFFER_POOL { PRIMARY | ALTERNATE } }
| RENAME { table_name TO new_table_name
           | COLUMN column_name TO new_column_name
           | INDEX index_name TO new_index_name }
};
```

In the following example, the ALTER TABLE statement is used to define an encryption policy for a table:

```
ALTER TABLE pub.customer SET ENCRYPT WITH 'AES_CBC_192';
```

Using the REKEY option, a security administrator or DBA can re-encrypt the existing object policy with a new key using the same cipher. When this option is used, SQL writes a new object policy with the same cipher name to the security schema.

Specifying SET DECRYPT creates an object policy with null values, indicating that the data will not be encrypted for the database object.

Viewing encryption policy with the SHOW ENCRYPT statement

The SHOW ENCRYPT statement provides encryption policy information on the primary database. It can be used only by security administrators or DBAs. It uses the following syntax:

```
SHOW ENCRYPT ON { ALL [ TABLE | INDEX | LOB ]
                | TABLE table_name [ WITH INDEX | WITH LOB ]
                | TABLE table_name ON INDEX index_name }
;
```

When run, the statement returns a result set with eight columns:

- Database object type (AREA, TABLE, INDEX, LOB)
- Database object name
- Objects table name (blank for area)
- Database object name (blank for area)
- Database object identification
- Object policy state (CURRENT or PREVIOUS)
- Object policy cipher name
- Object policy version number

Only active policies are returned by the statement. The only option which shows Type I area encryption information is the SQL statement SHOW ENCRYPTION ON ALL. Other options on SHOW ENCRYPT show encryption information only for Type II area database objects.

Deleting encrypted objects with OpenEdge SQL

To delete an encrypted object with OpenEdge SQL, use the DROP TABLE, DROP INDEX, or ALTER TABLE DROP COLUMN statements. When any of the statements are invoked, OpenEdge SQL removes any policies in the database security schema that are associated with the object to be deleted.

Disabling encryption

If you determine that you no longer need your database encrypted, you can disable encryption with the PROUTIL DISABLEENCRYPTION command. PROUTIL DISABLEENCRYPTION requires ABL security administrator or SQL database administrator privileges and key store admin privileges.The basic syntax is:

```
proutil dbname -C disableencryption
```

Disabling encryption decrypts all the data in your database, removes all the encryption policies, and archives your key store (by renaming it to *dbname*.ksbk).

If you have a large number of encrypted database objects, you might want to control the decryption of your objects prior to executing the DISABLEENCRYPTION command. To decrypt your data, update the encryption policy to the NULL cipher.

To decrypt data:

1. First verify that the object is encrypted with EPOLICY SCAN, as shown:

```
proenv>proutil t1demo -C epolicy scan area "dataarea101"
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
AREA     dataarea101 / 101  CURRENT   AES_CBC_128  V:1   627 of 627 blocks encrypted
```

2. Change the encryption policy cipher to the NULL cipher (cipher 0), as shown:

```
proenv>proutil t1demo -C epolicy manage area cipher "dataarea101" -Cipher 0
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
Encryption policy cipher change for Area dataarea101 in Area 101
Cipher specification change to NULL_NULL_NULL completed.
```

3. You can re-confirm the policy change with EPOLICY SCAN, as shown:

```
proenv>proutil t1demo -C epolicy scan area "dataarea101"
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
AREA  dataarea101 / 101 CURRENT NULL_NULL_NULL V:2 1 of 627 blocks encrypted
AREA  dataarea101 / 101 PREVIOUS AES_CBC_128 V:1 626 of 627 blocks encrypted
```

4. Update all the blocks to the NULL cipher policy with EPOLICY MANAGE UPDATE, as shown:

```
proenv>proutil t1demo -C epolicy manage area update "dataarea101"
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
```

5. Confirm that the object is decrypted and the policy deleted with EPOLICY SCAN, as shown:

```
proenv>proutil t1demo -C epolicy scan area "dataarea101"
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
No encryption policy exists for object dataarea101.
EPolicy: Encryption policy management failed -402
```

6. Disable encryption, as shown:

```
proenv>proutil t1demo -C disableencryption
OpenEdge Release 10.2B1P as of Fri Nov 20 19:01:52 EST 2009
Updating blocks of object 9/7. (15466)
Updating blocks of object 9/100. (15466)
BI Encryption has been disabled for database t1demo. (12490)
Encryption Feature has been disabled for database t1demo. (12490)
The BI file is being automatically truncated. (1526)
```

By decrypting your data prior to disabling encryption, you can control the impact of the update to normal database processing.

You can cancel DISALBEENCRYPTION at anytime during the execution of the command. Cancelling DISABLEENCRYPTION does not affect the integrity and usability of the database, however cancelling does not revert the changes that have been made. Cancelling DISABLEENCRYPTION may leave the database in one the following states:

- BI encryption is disabled (in single-user mode if it was enabled before the disablement)
- AI encryption is disabled
- All ciphers of current policies are set to null-null-null
- All data blocked are decrypted
- The encryption feature is fully disabled
- All encryption policies are removed

Check your database `.lg` file or the status messages displayed on screen to determine the state of the database.

A

Audit Data Tables

When you audit-enable a database, the audit tables dedicated to auditing support are automatically created in the enabled database. This appendix provides a description of each table and its fields.

For details, see the following topics:

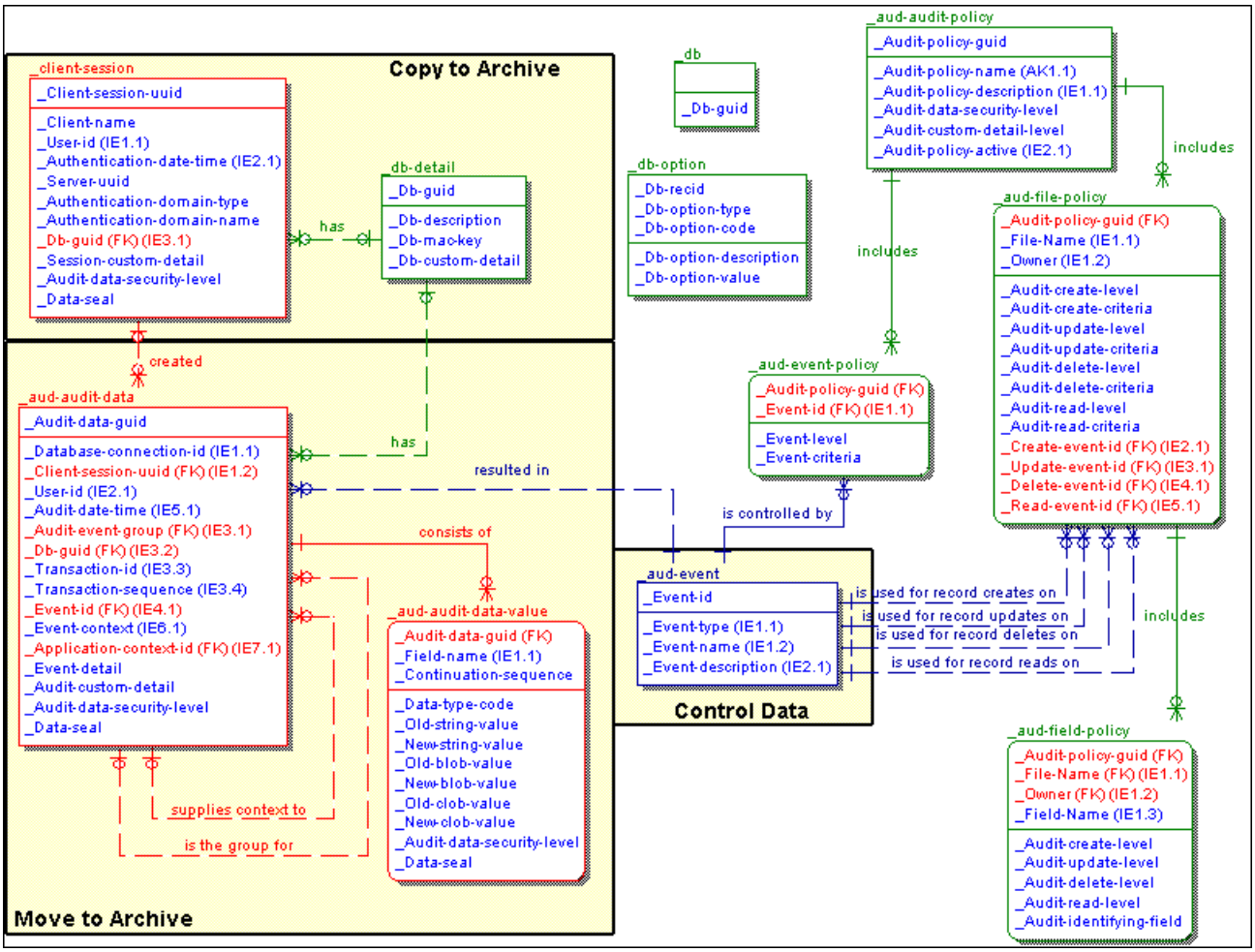
- Audit data table schema

Audit data table schema

You cannot modify the schema of the audit tables. You can add a child table to any of the audit tables (by using a foreign key relationship to a record field in the existing audit data table); however, there will be no archiving or security extended to the child table.

The following figure illustrates the auditing metadata.

Figure 13: OpenEdge auditing metadata



The audit tables fall into two categories: those that hold the audit data and those that hold the audit policy configuration.

The following tables hold audit data:

- `_aud-audit-data`
- `_aud-audit-data-value`

The following tables hold audit policy configuration:

- `_aud-audit-policy`
- `_aud-event`
- `_aud-event-policy`
- `_aud-field-policy`
- `_aud-file-policy`

The following tables support auditing specifically or support auditing and other functionality:

- **`client-session`** — Records additional login session information that further defines the login domain authentication system

- **_db-detail** — Records the binding between a databases identity (UUID) and the encrypted MAC key used for audit data and archive integrity
- **_db-option** — Records the database and application options for controlling database and application operations such as auditing and security

In addition, the existing _db table (which contains database-wide information) has been modified for auditing.

The following table lists and describes all of the audit tables:

Table 15: OpenEdge audit tables

Table name	Description
aud-audit-data	The actual audit data created as a result of enabled audit events according to the audit policy. All audit events, including both database and application-level audit events, are stored in this central table. However, certain fields are specific to database auditing events, and the purpose and use of some of the fields can vary depending on the type of audit event. This table schema is used for both the online audit data as well as the archive audit data. For nondatabase (create, update, delete) auditing, the fields that sequence the order of the audit data can be used for the application. For example, the sequence fields can be used to keep an ordered count of failed login attempts, or perhaps the sequence of tasks within a workflow. To avoid tampering of the data, the contents of this table can be deleted only by the registered audit data archiver. The data in this table can be sealed with a Message Digest or MAC for nonrepudiation. There are a number of indexes required on this table to facilitate efficient reporting. These indexes can be made inactive in the operational database and enabled only in an archive database, if required to improve performance of queries. Records in this table can be moved to an archive database as part of the archive process.

Table name	Description
_aud-audit-data-value	A child table for the _aud-audit-data table; contains a record for each modified field value. A separate record is created for each modified field to make it easier to report on changes made to a specific field over time or to hold field values of large size. Whether just new values, or both old and new values, are stored is driven by the audit policy levels defined on the _file and _field tables. Values are stored in strings, and the string values are always stored in American format using the databases code page so that they are consistent. In order to remove the dependency on the originating database, the data type and name of the field as it was at the time the value was audited is recorded in this table. This table provides for continuation records to support large data that might exceed record limits. To avoid tampering of the data, the contents of this table can be deleted only by the registered audit data archiver. The data in this table can be sealed with a Message Digest or MAC for nonrepudiation. Records in this table can be moved to an archive database as part of the archive process.
_aud-audit-policy	Provides a mechanism to define named audit policies that contain sets of audit configuration data according to different policy requirements, such as comprehensive auditing, standard auditing, minimum auditing, or secure auditing, for example. An audit policy controls application, database, and internal auditing settings. The use of named audit policies allows standard or template policies to be defined and deployed to target databases. As the audit policy is not tied to a particular database GUID, changes to the database GUID (for example, through dump or load, or database copies) do not impact the audit policy. It is possible to have multiple policies active; in this case, the aggregation of the policies then applies. If there are conflicts across the multiple active policies, the highest level of auditing security applies. Audit policy information is not copied to the archive database. However, modifications to the policy and information about which policies are active must be audited, and these audit entries become part of the archive.
_aud-event	Identifies each defined audit event and its corresponding event ID, which is a binary (integer) code used on the audit data table to save space. The event ID will also be referenced in code when creating audit data. You cannot import policies that contain events not defined in this table. Operational code does not require the details in this table, as it makes reference directly to the event ID. These details are used primarily for reporting and documentation and can also be used during the archive process to stored the archived audit data with more meaningful event names, if required. Note that this event data will be required in every database that has auditing enabled. If additional events are created, these will also have to be updated into all auditing databases.

Table name	Description
	The fields used to control whether the event is enabled or not and the level of detail to capture are defined in the audit policy tables. The table facilitates the logical grouping of events by type and makes provisions for a detailed description of the event and its use in relation to auditing. This grouping mechanism can prove useful to simplify the tools used to maintain the audit functionality.OpenEdge is preconfigured with standard OpenEdge-defined events that have specific event IDs; however, developers can extend the supported events as required by their application. All event IDs up to 32000 are reserved for use by OpenEdge. The following event types are available: • **Admin** — Database started or stopped • **Application** — Client authentication, login, logout, reauthenticate, authorization to resources, for example • **Audit** — Database auditing enabled or disabled; or an audit policy record created, updated, or deleted • **Data** — Create, update, or delete events • **Schema** — Schema changes • **Security** — User account or authentication domain created, updated, or deleted, for example • **User** — User login success, user logout, or user login failed; or database user identity set or failed, for example • **Utility** — Database dump or load, or database table move, for example Since OpenEdge reserves all event IDs up to 32000 for internal use, application-specific events must use an ID greater than 32000. The data in this table is copied to each database that has auditing enabled, including the archive databases.
`_aud-event-policy`	Defines the event ID and the level of detail to be recorded for those audit events used within a policy. Note that the policy is set globally for the event. Audit policy information is not copied to the archive database; however, modifications to the policy must be audited. These audit entries will be part of the archive.
`_aud-field-policy`	Defines field-level auditing configuration settings for a named audit policy. Records will exist only for fields that have specific auditing policy settings. If a record does not exist for a field, then, by default, auditing for the field is the same as what is defined for the table to which the field belongs. If no table settings are defined, auditing is, by default, disabled for the field. You can enable or disable auditing of data for specific fields and control the level of detail kept for each field for each auditable database event. Audit policy information is not copied to the archive database, but modifications to the policy must be audited. These audit entries will be part of the archive.

Table name	Description
_aud-file-policy	Defines file- (table-) level auditing configuration settings for a named audit policy. Records will exist only for tables that have specific auditing policy settings. If a record does not exist for a table, then, by default, auditing is disabled for that table. You can enable or disable auditing of data for a table or its specific fields, determine what events to audit for the table, and select the level of audit detail to keep in the audit data table for the specified events. Audit policy information is not copied to the archive database, but modifications to the policy must be audited. These audit entries will be part of the archive.
_client-session	Records information about a client-authenticated login session. The table stores information about the client session and the methods used to authenticate the client. The tables primary purpose is to provide additional information for auditing for the purposes of nonrepudiation—to provide extra details as to exactly who the user was. This table is part of a blank, empty database, as it has other uses beyond auditing. When used for auditing, the client session UUID will be used to associate the authenticated client session with the audit events. An authenticated client session might not always be available when recording auditing events; an example of this occurs when an OpenEdge AppServer™ connects to a database outside of the context of an authenticated session. The data in this table can be sealed with a Message Digest or MAC for nonrepudiation. Note that the data in this table must be copied to the archive database as part of the archive process, but the contents will not be deleted as part of the same process since the client session information could be used for purposes other than auditing.
_db	This is the existing _db metaschema table for databases. Added to the table is a new field for a globally unique identifier (GUID) to uniquely identify a database and support the aggregation of audit data from multiple databases or sources. Note that the database GUID can change, for example, through backup/restore and copy operations to new databases. Therefore, give careful consideration when using it. The foreign key to related tables as joins could be broken or might need to be fixed, depending on the nature of the related data.

Table name	Description
_db-option	Provides an extension to the _db database table for additional database options (user-defined fields). This is a generic table designed to make the _db metadata extensible. The use of option types up to 32000 are reserved by OpenEdge for internal use. An example entry in this table would be for a database option that controls whether or not to record authenticated client sessions in the _client-session table.
_db-detail	An extension to the _db database table to hold auditing-specific information. The join to _db is on the _Db-guid foreign key. For example, the primary reason to use an additional table for the auditing information is to facilitate multiple record entries to accommodate different database GUIDs, such as when copying the contents of this table to an archive database for long-term storage where the archive database contains audit data from multiple databases. Note that the join from the _db table is not physically shown because this data is copied with the archive process and, therefore, the record in the _db table might be for a different database, breaking the referential integrity. The details in this table provide a meaningful description of the database associated with a GUID and, as this is the only means to describe what the database is when the audit data contains entries for multiple databases, the contents of this table must be copied with the audit data. Additionally, this table records the MAC key used to secure and seal the audit data. Without the MAC key, the data in the audit table cannot be validated, and so the MAC key must be copied with the audit data. If the MAC key is ever modified for a particular database, then a new database GUID must be created, since a database GUID can only have a single MAC key. It is the combination of the database GUID and the MAC key that facilitates access to sealed audit data. Also, if the database GUID ever changes, a new record must be created in this table using the current MAC key and a meaningful description of the database associated with the GUID. The contents of this table must be copied as part of the archive process, as the data is needed to describe the database and unseal the audit data. The contents, however, must not be deleted as part of the archive process, only copied to the archive database.

The following tables describe each of the audit tables individually.

Table 16: _aud-audit-data table

Field name	Description	Data type	Format
_Audit-data-guid	A globally unique ID that identifies the audit data record. The ID must be globally unique to support the aggregation of audit data across multiple databases and, potentially, applications. The primary purpose of this unique ID is to provide a foreign key for the child table _aud-audit-data-value that is used to store individual, modified field values for database events. The value is always a 22-character, BASE-64 encoded UUID.	CHARACTER	X(28)
_Database-connection-id	For database connections, this is the ID that is used to track audit events for the duration of a specific database connection. This is relevant only to database create, update, and delete events.	CHARACTER	X(28)
_Client-session-uuid	A unique ID for the client login session; used to tie together all the audit data for a specific session. The value is always a 22-character, BASE-64 encoded UUID. For OpenEdge AppServer sessions or connections that do not have a valid authenticated client session record in the _aud-client-session table, this field would refer to the UUID of the AppServer. This field is not used by either SETUSERID( ) or the -U and -P parameters. This field is indexed together with the database connection ID field, so to obtain the details for a specific session, the database connection information must also be specified for efficiency. Multiple client sessions could occur within a single database connection.	CHARACTER	X(28)
_User-id	The authenticated user who has been authorized to perform a database operation or application function. This user ID can also include the domain the user was authenticated by. For example: fred@domain This field is copied from the _client-session table for ease of reference or querying and also because the use of the client session is optional.	CHARACTER	X(35)

Field name	Description	Data type	Format
_Audit-date-time	The date and time that the audit event occurred stored as date time timezone, which results in the time recorded to the millisecond.	DATE	99/99/9999
_Audit-event-group	A GUID pointing at the audit data record containing the audit event group events details. The value is always a 22-character, BASE-64 encoded UUID. Within an application context there can exist zero or more optional audit event groups that can carry information such as the business entity, business object, or data object. Audit event groups have starting and ending events that can be used to establish a collection of related database audit events. For example, starting an update on a ProDataSet as part of a business task might change multiple table records in multiple databases. By starting and ending an audit event group around the ProDataSet UPDATE operation, you can later audit all audit event records that were involved in that single application operation. Where an audit event group is used, its ID will point at an _aud-audit-data record that describes the details of the audit event group, who started it, when, etc. This field will therefore contain an _Audit-data-guid value as a foreign key. Starting an audit event group will create an audit data entry for the audit event group and propagate this unique ID to all connected databases with auditing enabled so they can use this in their _Audit-event-group field, until it is reset. This facilitates storing extra information about the audit event group without duplicating all the details onto every audit data record.	CHARACTER	X(28)

Field name	Description	Data type	Format
	Note the following: • Audit event groups are used for auditing purposes only and do not support roll-back of data manipulation operations. The ending of an audit event group does not happen automatically, even at program exit. The application programmer must be careful to end an audit event group unless it can be assumed to include all of the audit events up to the end of the program's execution. • For the audit data record that stores the audit event group details, the `_audit-event-group` field will be blank. If any structure or subgrouping is required for audit event groups, then the `_Event-context` can be used to provide this.		
`_Db-guid`	A globally unique identifier (GUID) for the database to support the aggregation of data across multiple databases. The value is always a 22-character, BASE-64 encoded UUID. This field is used in all audit-related tables to identify which database the audit information was recorded in. All other fields are then unique only within a specific database. **Note:** The database GUID might change, for example, through backup and restore or copy operations to new databases. Therefore, give careful consideration when using it. The foreign key to related tables as joins could be broken or might need to be fixed, depending on the nature of the related data. The other way to join from the `_db` table is through the `RECID`, which never changes but is also not globally unique; which option to use depends on the related data.	`CHARACTER`	X(28)
`_Transaction-id`	The database transaction ID for database events.For nondatabase events, this could be an application-supplied transaction sequence, such as a task ID. **Note:** Transaction IDs are client-specific and are not unique in the database. The IDs can be reset and do not necessarily guarantee the order of the audit data.	`INTEGER`	->>>>>>>9

Field name	Description	Data type	Format
_Transaction-sequence	The sequence of updates within a transaction ID. The sequence is incremented only for auditable events so that there are no holes in the sequence. This field allows for a finer level of granularity than the audit date and time and, therefore, supports multiple updates within a millisecond while still being able to determine the order in which the updates occurred.	INTEGER	->>>>>>>9
_Event-id	A unique identifier for the event. This is the binary code for the event that is stored in the audit data table and referenced in code. OpenEdge includes a standard set of supported event IDs, but the list can be extended by developers as required. A separate table, _aud-event, exists to define what the event IDs mean or are used for and whether the event is enabled through the audit policy tables.	INTEGER	->>>>>>>9
_Event-context	The context within which the event occurred. For application events, this could be a particular procedure or function or an action a user was authorized to access. For database create, update, or delete events, this could simply be the table name or the database.table name followed by a delimited list of identifying field values to identify the record that was modified. For example: database.table + CHR(6) + field1value + CHR(7) + field2value + CHR(7) + field3value The control code CHR(6) denotes the start of the identifying field value part of the context, and the CHR(7) is used to delimit the field values where a record is identified by more than one field. Note the following: • Identifying fields values can also be stored in the child table _aud-audit-data-value if the value was modified and the policy is configured to audit old and new values for these fields. This field does not show old and new values because it is used to locate audit changes for a specific record in a table. • For database events, the execution context will be written as a separate record with an event name of pvm.appcontext. This is to avoid	CHARACTER	X(3000)

Field name	Description	Data type	Format
	repeating the execution context on every audit record, writing new context details only when the execution context changes. The field _Application-context-id points to the record that contains the actual context.		
_Application-context-id	The application context of an audit log entry is stored in a separate _aud-audit-data record with full details of the context, such as when it was created or by whom, for example. A special event ID is used to identify context records. The content of the context data is user-defined, and the current context is set and reset using audit context ABL statements. The context record is stored in the database pointed at by the AUDIT-DB alias. The key to the context record is then propagated to all connected databases that have auditing enabled, and this key is written into this field as a foreign key reference to the full context data, until it is reset. This avoids duplicating large amounts of context data on every audit data record, yet still provides an efficient way to retrieve the context from the separate context record. **Note:** The details of the context data will be in the databases that have the context event ID enabled.	CHARACTER	X(28)
_Event-detail	The detail for the event. The detail can be a stream of field values that are set to be recorded in compressed field-value mode. For database events, this would be blank, as the details of what changed are stored in the child table _aud-audit-data-value with a record for each modified field value. For application events, its use is user-defined.	CHARACTER	X(3000)
_Audit-custom-detail	A custom field provided to support additional audit data required by an application.	CHARACTER	X(3000)

Field name	Description	Data type	Format
_Audit-data-security-level	This field is copied from the _Audit-data-security-level field on the _aud-audit-policy table and records the data security level used for this record. The value of this field determines what information is written to the _Data-seal field at creation time as follows: • **0** — No additional security applied to audit data. • **1** — Stores a message digest in the _Data-seal field of the audit data tables to guarantee the integrity of the data. • **2** — Stores a MAC in the _Data-seal field of the audit data tables to guarantee the integrity of the data. A MAC is more secure than a message digest. The MAC key to use is stored in the associated field _Audit-mac-key. The field value is used when validating audit record seals to indicate what validation method to use.	INTEGER	->9
_Data-seal	An internally generated data integrity seal on the data. The seal guarantees the integrity of the data and ensures it has not been tampered with outside of the registered services permitted to maintain this data. This could be a message digest or a MAC, depending on the policys security level at the time of creation. The use of the data seal is optional. It guarantees nonrepudiation of the data but will have a cost of additional storage space and a possible impact on performance to assign a value and check the results.	CHARACTER	X(28)

Table 17: _aud-audit-data-value table

Field name	Description	Data type	Format
_Audit-data-guid	A globally unique ID that identifies the audit data record. The value is always a 22-character, BASE-64 encoded UUID. The ID must be globally unique to support the aggregation of audit data across multiple databases and, potentially, applications. The primary purpose of this unique ID is to provide a foreign key for the child table _aud-audit-data-value that is used to store individual, modified field values for database events.	CHARACTER	X(28)
_Field-name	The actual name of the field the audit value is for, such as customerName, for example.	CHARACTER	X(32)
_Continuation-sequence	The continuation sequence facilitates breaking a single audit record value into multiple parts to avoid breaking record limits where the audit data value is too large to fit into a single record. A possible reason for this is because the field name, old value, and new value are being stored; and one of the values itself could, in fact, exceed the record limit of 32K.	INTEGER	->9
_Data-type-code	Contains an integer code for the data type of the audit value where the audit value is stored as a string. A code for the data type is used to minimize the storage impact of recording this information. The primary reason for recording the data type of the audit value is to guard against schema changes, that is, to cater for data type changes, and to remove the dependency on the originating database to determine the data type of the value. The data type is required for reporting in order to display the value in the appropriate format, as the actual value is stored as a character string in a consistent format (American). The value must be cast to the originating native data type in order to deal with internationalization formatting issues. Sample data type codes are: • **1** — Character • **2** — Date	INTEGER	->9

Field name	Description	Data type	Format
	• **3** — Logical • **4** — Integer • **5** — Decimal • **7** — Recid • **8** — Raw • **11**— Memptr • **13** — Rowid • **18** — Blob • **19** — Clob • **34** — Datetime • **39** — Longchar • **40** — Datetime-TZ		
`_Old-string-value`	If the field can be represented as a string value, then this field contains the old string value of the modified field. This value is in American format for consistency.	`CHARACTER`	X(3000)
`_New-string-value`	If the field can be represented as a string value, then this field contains the new string value of the modified field. This value is in American format for consistency.	`CHARACTER`	X(3000)

Field name	Description	Data type	Format
_Audit-data-security-level	This field is copied from the _Audit-data-security-level field on the _aud-audit-policy table and records the data security level used for this record at the time the record was created. The value of this field determines what information is written to the _Data-seal field: • **0** — No additional security applied to audit data. • **1** — Stores a message digest in the _Data-seal field of the audit data tables to guarantee the integrity of the data. • **2** — Stores a MAC in the _Data-seal field of the audit data tables to guarantee the integrity of the data. A MAC is more secure than a message digest. The MAC key to use is stored in the associated field _Audit-mac-key.	INTEGER	->9
_Data-seal	An internally generated data integrity seal on the data that guarantees the integrity of the data and ensures it has not been tampered with outside of the registered services permitted to maintain this data. This could be a message digest or a MAC. The use of the data seal is optional, as it guarantees nonrepudiation of the data but will have a cost of additional storage space and a possible impact on performance to assign a value or check the results.	CHARACTER	X(28)

Table 18: _aud-audit-policy table

Field name	Description	Data type	Format
_Audit-policy-guid	A globally unique ID used to identify the audit policy. The value is always a 22-character, BASE-64 encoded UUID. The ID is used as the foreign key on policy-related tables. The use of a unique ID for the foreign key facilitates changes to the audit policy details, such as the name.	CHARACTER	X(28)
_Audit-policy-name	A unique name used to identify the audit policy. As the name is not used as a foreign key to related tables, changes to the policy name are permitted.	CHARACTER	X(35)

Field name	Description	Data type	Format
_Audit-policy-description	A free text description used to describe the purpose or use of the audit policy.	CHARACTER	X(70)
_Audit-data-security-level	A database-wide field that controls the level of security applied to the audit data transaction tables _client-session, _aud-audit-data, and _aud-audit-data-value, each of which has a field called _Data-seal. The setting of this field determines what information is written to the _Data-seal field, as follows: • **0** — No additional security applied to audit data. • **1** — Stores a message digest in the _Data-seal field of the audit data tables to guarantee the integrity of the data. • **2** — Stores a MAC in the _Data-seal field of the audit data tables to guarantee the integrity of the data. A MAC is more secure than a message digest. The MAC key to use is stored in the associated field _Audit-mac-key. If this field value is ever changed, it will impact how the audit data is sealed or unsealed; therefore, the value must be recorded along with the _data-seal in each table that is sealable.	INTEGER	->9

Field name	Description	Data type	Format
_Audit-custom-detail-level	Applicable only to application-audit events (not database tables and fields); controls whether to record the value in the _Audit-custom-detail field on the _aud-audit-data record, as follows: • **0** / **OFF** — Ignores audit custom detail parameter • **1** /**ON** — Records passed-in audit custom detail value in the _Audit-custom-detail field Note that within a single database with multiple active policies, multiple custom detail levels can be active, as each policy can have a different custom detail level. The custom detail level to use in each case will be the maximum value that applies.	INTEGER	->9
_Audit-policy-active	A flag (set to **NO** by default) that determines which audit policies are active for the current database. It is possible to have multiple policies active, and the aggregation of the policies then applies. If there are conflicts across the multiple active policies, then the highest level will apply. Changing which policies are active is an auditable event.	LOGICAL	YES/NO

Table 19: _aud-event table

Field name	Description	Data type	Format
_Event-id	A unique identifier for the event. This is the binary code for the event that is stored in the audit data table and referenced in code. OpenEdge ships with a standard set of event IDs supported, but the list can be extended by developers as required. OpenEdge reserves all event IDs up to 32000.	INTEGER	->>>>>9
_Event-type	This is the type of audit event used to categorize the events for tools and reporting purposes. This field has no meaning in operational code.The following event types are available: • **Admin** — Database started or stopped	CHARACTER	X(15)

Field name	Description	Data type	Format
	• **Application** — Client authentication, login, logout, reauthenticate, authorization to resources, for example • **Audit** — Database auditing enabled or disabled; or an audit policy record created, updated, or deleted • **Data** — Creates, updates, or deletes events • **Schema** — Schema changes • **Security** — User account or authentication domain created, updated, or deleted, for example • **User** — User login success, user logout, or user login failed; or database user identity set or failed, for example • **Utility** — Database dump or load, or database table move, for example		
`_Event-name`	This is a meaningful name for the event within the context of the event type. The event type and the event name together describe the event and ideally should be unique.	`CHARACTER`	X(35)
`_Event-description`	This is a free text description of the event used to explain the purpose of the audit event with sample use cases, etc.	`CHARACTER`	X(500)

Table 20: _aud-event-policy table

Field name	Description	Data type	Format
`_Audit-policy-guid`	This is a globally unique ID used to identify the audit policy. The value is always a 22-character, BASE-64 encoded UUID. The GUID is used as the foreign key on policy-related tables. The use of a unique ID for the foreign key facilitates changes to the audit policy details, such as the name.	`CHARACTER`	X(28)

Field name	Description	Data type	Format
_Event-id	A unique identifier for the event. This is the binary code for the event that is stored in the audit data table and referenced in code. OpenEdge ships with a standard set of event IDs supported, but the list can be extended by developers as required. OpenEdge Release reserves all event IDs up to 32000.	INTEGER	->>>>>9
_Event-level	This field controls whether auditing for this event is enabled or not, and, if auditing is enabled, the level of detail to capture for the event. The supported values are as follows: • **0** — Auditing off for this event (default) • **1** — Auditing on for this event; records minimum details • **2** — Auditing on for this event; records full details If this field has a value greater than **0**, then auditing is enabled for this event. The meaning of minimum or full details is dependent on the type and nature of the event. For example, for a database connection event, minimum details could simply record that the connection was made and by whom, whereas full details could include the physical connection parameters used. The use of full details is optional and might not always be relevant to all event types.	INTEGER	->9

Table 21: _aud-field-policy table

Field name	Description	Data type	Format
_Audit-policy-guid	This is a globally unique ID used to identify the audit policy. The value is always a 22-character, BASE-64 encoded UUID. The ID is used as the foreign key on policy-related tables. The use of a unique ID for the foreign key facilitates changes to the audit policy details, such as the name.	CHARACTER	X(28)

Field name	Description	Data type	Format
_File-Name	This is the actual table name of the database table that the audit policy settings are for.	CHARACTER	X(32)
_Owner	The schema owner for the table, such as PUB, for example. SQL allows for the same table in different schemas.	CHARACTER	X(32)
_Field-Name	This is the actual field name of the database field that the audit policy settings are for.	CHARACTER	X(32)
_Audit-create-level	This field defines the level of auditing active for this specific field during a create event. If create auditing is turned off at the table level (the _Audit-create-level in the _aud-file-policy table is set to **0**), then this field is ignored, as shown: • **–1** —Turns create auditing off for this field. **Do not record any record-change events or field values.** • **0** — Uses the audit level setting as defined by the _Audit-create-level in the _aud-file-policy table; do not override it at the field level. This is the default. • **1** — Min Audit Data. Stores only when the audit event occurred and who caused the event but does not store additional details for the initial values. • **2** — Standard Audit Data. Same as **1** but additionally stores the initial values in the audit data. • **12** — Records both record-change events and field values in default (streaming) mode, if possible. When create auditing is done, audit data would typically not include values, as these would simply indicate the initial values. For creates, an update event would also occur where the values could be recorded.	INTEGER	->9
_Audit-update-level	This field defines the level of auditing active for this specific field during an update event. If update auditing is turned	INTEGER	->9

Field name	Description	Data type	Format
	off at the table level (`_Audit-update-level` in the `_aud-file-policy` table is set to **0**), then this field is ignored, as shown: • **–1 —** Turns off update auditing for this field. **Does not record any record-change events or field values.** • **0** — Uses the audit level setting as defined by the `_Audit-update-level` in the `_aud-file-policy` table; does not override it at the field level. This is the default. • **1** — Min Audit Data. Stores only when the audit event occurred and who caused the event but does not store additional detail for the before and after values of this field. • **2** — Standard Audit Data. Same as **1** but additionally stores the new value in the audit data where applicable. In order to compare old and new values, previous records must be read to determine the old value and rely on the correct sequence of the audit events in the audit data table. This cannot always be guaranteed. Note that only modified data is kept in the audit data table, so many records might need to be read to get to the old value for a specific field. • **3** — Full Audit Data. This is the same as **2** but additionally stores both the old and new values in the audit data table. This makes it easier to query or compare old and new values but requires additional storage space for the audit data. • **12** — Records both record-change events and the new value of only the changed fields in default (streaming) mode, if possible. • **13** — Records both record-change events and the new and old values of only the changed fields in the default (streamed) mode, if possible.		

Field name	Description	Data type	Format
_Audit-delete-level	This field defines the level of auditing active for this specific field during a delete event. If delete auditing is turned off at the table level (the _Audit-delete-level in the _aud-file-policy table is set to **0**), then this field is ignored, as shown: • **–1** —Turns off delete auditing for this field. **Does not record any record-change events or field values.** • **0** — Uses the audit level setting as defined by the _Audit-delete-level in the _aud-file-policy table; does not override it at the field level. This is the default. • **1** — Min Audit Data. Stores only when the audit event occurred and who caused the event but does not store additional details for the final field value. • **2** — Standard Audit Data. Same as **1** but additionally stores the final field value in the audit data to record the final state of the record prior to deletion. • **12** — Records both record-change events and field values in default (streamed) mode, if possible.	INTEGER	->9
_Audit-identifying- field	This identifies a field as being an identifying field that would be used to find a unique record in the table. It is an integer to support multiple identifying fields in a specified sequence. Typically, this would be the fields that make up the primary index on the table, but it could be something else. Therefore, it is configurable. Fields that are made identifying fields by specifying a value greater than **0** are then stored in the _event-context on the audit data table to determine which record actually changed. If no fields are marked as identifying fields, then the fields in the primary index would be used by default.	INTEGER	->9

Table 22: _aud-file-policy table

Field name	Description	Data type	Format
_Audit-policy-guid	A globally unique ID used to identify the audit policy. The value is always a 22-character, BASE-64 encoded UUID. The ID is used as the foreign key on policy-related tables. The use of a unique ID for the foreign key facilitates changes to the audit policy details, such as the name.	CHARACTER	X(28)
_File-Name	The actual name of the database table that the audit policy settings are for.	CHARACTER	X(32)
_Owner	The schema owner for the table, such as PUB, for example. SQL allows for the same table in different schemas.	CHARACTER	X(32)
_Audit-create-level	The default level of auditing active for the table for create events. The level can have one of the following values: • **0** — Auditing is off for this table for create events. This is the default. If this field is **0** at the table level, then any more specific audit settings on the _aud-field-policy table are ignored. • **1** — Min Audit Data. Stores only when the audit create event occurred and who caused the event but does not store additional details for the initial values. • **2** — Standard Audit Data. Same as **1** but additionally stores the initial values in the audit data. • **12** — Records both record-change events and field values in default (streamed) mode, if possible. If auditing is enabled by specifying an audit level greater than **0**, then the level of detail recorded can be overridden at the field level using _Audit-create-level set on each _aud-field-policy record.	INTEGER	->9
_Audit-update-level	Defines the default level of auditing active for the table for update events. The level can have one of the following values:	INTEGER	->9

Field name	Description	Data type	Format
	• **0** — Auditing is off for this table for update events. This is the default. If this field is **0** at the table level, then any more specific audit settings on the `_aud-field-policy` table are ignored. • **1** — Min Audit Data. Stores only when the audit update event occurred and who caused the event but does not store additional details for the before and after values. • **2** -— Standard Audit Data. Same as **1** but additionally stores the new value in the audit data where applicable. In order to compare old and new values, previous records must be read to determine the old value and rely on the correct sequence of the audit events in the audit data table. This cannot always be guaranteed. Note that only modified data is kept in the audit data table, so many records might need to be read to get to the old value for a specific field. • **3** — Full Audit Data. This is the same as **2** but additionally stores both the old and new values in the audit data table. This makes it easier to query or compare old and new values but requires additional storage space for the audit data. • **12** — Records both record-change events and the new value of only the changed fields in default (streamed) mode, if possible. • **13** — Records both record-change events and the new and old values of only the changed fields in default (streamed) mode, if possible. If auditing is enabled by specifying an audit level greater than **0**, then whether an audit record is created in the audit data table is further controlled by the `_Audit-update-level` on the `_aud-field-policy` records.		
`_Audit-delete-level`	Defines the default level of auditing active for the table for delete events. The	`INTEGER`	->9

Field name	Description	Data type	Format
	level can have one of the following values: • **0** — Auditing is off for this table for delete events. This is the default. If this field is **0** at the table level, then any more specific audit settings on the `_aud-field-policy` table are ignored. • **1** — Min Audit Data. Stores only when the audit delete event occurred and who caused the event but does not store additional details for the actual values. • **2** — Standard Audit Data. Same as **1** but additionally stores the current values in the audit data to record the final state of the record before it was deleted. • **12** — Records both record-change events and field values in default (streamed) mode, if possible. If auditing is enabled by specifying an audit level greater than **0**, the level of detail recorded can be overridden at the field level using `_Audit-delete-level` set on each `_aud-field-policy` record.		
`_Create-event-id`	The event ID to use for record creates for this table. The default is 5100 for normal record creates, but this can be set to a different value for grouping of related create events across multiple tables. For example, the create event ID could be directed to a specific event ID for creation of orders for all the tables related to order creation, such as order, order line, or stock tables, for example. This would make it easier to then query the audit data by event ID and report on all the data related to the order creation. This concept is used for internal system events to group things like schema changes, audit policy changes, or security permission changes, for example.	`INTEGER`	->>>>>9

Field name	Description	Data type	Format
	Note: It is invalid to have multiple active audit policies with the same table defined with conflicting event IDs.		

Field name	Description	Data type	Format
_Update-event-id	The event ID to use for record updates for this table. The default is 5101 for normal record updates, but the ID can be set to a different value for grouping of related update events across multiple tables. For example, the update event ID could be directed to a specific event ID for update of orders for all the tables related to order updates, such as order, order line, or stock tables, for example. This would make it easier to then query the audit data by event ID and report on all the data related to the order update. This concept is used for internal system events to group things like schema changes, audit policy changes, or security permission changes, for example. **Note:** It is invalid to have multiple active audit policies with the same table defined with conflicting event IDs.	INTEGER	->>>>>9
_Delete-event-id	The event ID to use for record deletes for this table. The default is 5102 for normal record deletes, but the ID can be set to a different value for grouping of related delete events across multiple tables. For example, the delete event ID could be directed to a specific event ID for deletion of orders for all the tables related to order deletion, such as order, order line, or stock tables, for example. This would make it easier to then query the audit data by event ID and report on all the data related to the order deletion. This concept is used for internal system events to group things like schema changes, audit policy changes, or security permission changes, for example. **Note:** It is invalid to have multiple active audit policies with the same table defined with conflicting event IDs.	INTEGER	->>>>>9

Table 23: _client-session table

Field name	Description	Data type	Format
_Client-session-uuid	A unique ID for the client login session and for when auditing is used to tie together all the audit data for a specific session. The value is always a 22-character, BASE-64 encoded UUID.	CHARACTER	X(28)
_Client-name	The name of the type of client session; for example: • **AS** — Appserver • **WS** — WebSpeed • **ABL** — ABL Client • **SQL** — SQL Client • **DBA** — Database Admin Tools	CHARACTER	X(35)
_User-id	The ID of the authenticated user.	CHARACTER	X(35)
_Authentication-date-time	The authentication date, time, and time zone.	DATE	99/99/9999
_Server-uuid	The unique ID for the connected server, such as OpenEdge AppServer or OpenEdge WebSpeed® server, to determine exactly which server was used when multiple servers are in operation.	CHARACTER	X(28)
_Authentication-domain-type	The type of authentication plug-in used, such as LDAP, for example.	CHARACTER	X(35)
_Authentication-domain-name	The application alias or logical name depending on the domain type or authentication plug-in being used; for example, for LDAP this would be an LDAP domain such as App1LDAP.	CHARACTER	X(70)
_Db-guid	A globally unique identifier (GUID) for the database where the authenticated client session was begun or originated. The value is always a 22-character, BASE-64 encoded UUID.	CHARACTER	X(28)
_Session-custom-detail	User-defined session information, such as terminal ID, provided in free text.	CHARACTER	X(3000)

Field name	Description	Data type	Format
_Audit-data-security-level	Records the data security level used for this record. This field is copied from the _Audit-data-security-level field on the _aud-audit-policy table. The value of this field determines what information is written to the _Data-seal field, as follows: • **0** — No additional security applied to audit data. • **1** — Stores a message digest in the _Data-seal field of the audit data tables to guarantee the integrity of the data. • **2** — Stores a MAC in the _Data-seal field of the audit data tables to guarantee the integrity of the data. A MAC is more secure than a message digest. The MAC key to use is stored in the associated field _Audit-mac-key.	INTEGER	->9
_Data-seal	An internally generated seal on the data that guarantees the integrity of the data and ensures it has not been tampered with outside of the registered services permitted to maintain this data. This could be a message digest or a MAC. The use of the data seal is optional, as it guarantees nonrepudiation of the data but will have a cost of additional storage space and a possible impact on performance to assign a value or check the results.	CHARACTER	X(28)

Table 24: _db table

Field name	Description	Data type	Format
_Db-guid	The current globally unique identifier (GUID) for the database to support the aggregation of data across multiple databases. The value is always a 22-character, BASE-64 encoded UUID. This field is used in all audit-related tables to identify the database for which the audit information exists. All other fields are then unique only within a specific database. **Note:** The database GUID can change, for example, through backup and restore or copy operations to new databases. Therefore, give careful consideration when using it. The foreign key to related tables as joins could be broken or might need to be fixed, depending on the nature of the related data. The other way to join from the _db table is through the RECID, which never changes but is also not globally unique; which option to use depends on the related data.	CHARACTER	X(28)

Table 25: _db-detail table

Field name	Description	Data type	Format
_Db-guid	A globally unique identifier (GUID) for the database to support the aggregation of data across multiple databases. The value is always a 22-character, BASE-64 encoded UUID. This field is used in all audit-related tables to identify the database for which the audit information exists. All other fields are then unique only within a specific database. **Note:** The database GUID can change, for example, through backup and restore and copy operations to new databases. Therefore, give careful consideration when using it. The foreign key to related tables as joins could be broken or might need to be fixed, depending on the nature of the related data. The other way to join from the _db table is through the RECID, which never changes but is also not globally unique; which option to use depends on the related data.	CHARACTER	X(28)

Field name	Description	Data type	Format
_Db-description	A meaningful description of the database associated with the current database GUID, used to describe the GUID where data is aggregated for multiple databases. The database physical name can be used to describe the database. If the database GUID ever changes, a new record will always be created in this table and the description must be reapplied to the new GUID.	CHARACTER	X(70)
_Db-mac-key	If the _Audit-data-security-level is set to **2** indicating that the security data is sealed using a MAC, then this field defines the MAC key to use. If the _Audit-data-security-level is not set to **2**, then this field will be ignored. Without the MAC key, the data in the audit table cannot be used; therefore, the MAC key must be copied with the audit data. If the MAC key is ever modified for a particular database, then a new database GUID must be created since a database GUID can only have a single MAC key. It is the combination of the database GUID and the MAC key that facilitates access to sealed audit data. Also, if the database GUID ever changes, a new record must be created in this table with the current MAC key and a meaningful description of the database associated with the GUID. **Note:** The MAC key must be stored as an encrypted raw value for security purposes, as it has characteristics similar to a password.	RAW	
_Db-custom-detail	A free-text field that provides additional custom information for a database specific to auditing; the detail will also be copied with the audit data as part of the archive process. **Note:** When the database GUID changes, a new record will be created in this table and the custom detail must be reapplied to the new record.	CHARACTER	X(3000)

Table 26: _db-option table

Field name	Description	Data type	Format
_Db-recid	The RECID of the _db table to identify the database for which these options exist. The RECID, rather than the database GUID, is used because the RECID never changes for a database. This information is specific to a single database and has no meaning outside of a database.	INTEGER	->>>>>>>9
_Db-option-type	The type of option (for example, security, auditing, or context) used to group related options together. OpenEdge reserves the first 32000 option types.	INTEGER	->>>>>9
_Db-option-code	A code within the option type to identify a specific option, such as recauthsession, to decide whether to record authentication sessions (yes or no), for example.	CHARACTER	X(35)
_Db-option-description	A free-text description of the database option.	CHARACTER	X(500)
_Db-option-value	The value of the database option; for example, for the option to record authenticated sessions, the value would be yes or no. This is a character field.	CHARACTER	X(70)

B

Preconfigured Audit Policies

OpenEdge includes several preconfigured audit policies that you can use as if or modify for your specific auditing needs.

For details, see the following topics:

- Using the preconfigured OpenEdge audit policies

Using the preconfigured OpenEdge audit policies

By importing the predefined OpenEdge database audit policies, you can audit the events listed in the following table. For more information, see the Audit Policy Maintenance Help.

Event ID	Event name	Event type	Event description	Policy name
100	`_sys.user.create`	Security	User account created	PSC-UserAccounts
101	`_sys.user.update`	Security	User account updated	PSC-UserAccounts
102	`_sys.user.delete`	Security	User account deleted	PSC-UserAccounts
210	`_sql.dba.create`	Security	SQL DBA created	PSC-Security

Event ID	Event name	Event type	Event description	Policy name
211	_sql.dba.update	Security	SQL DBA updated	PSC-Security
212	_sql.dba.delete	Security	SQL DBA deleted	PSC-Security
300	_sys.audit.policy.create	Audit	Audit policy record created	PSC-Audit
301	_sys.audit.policy.update	Audit	Audit policy record updated	PSC-Audit
302	_sys.audit.policy.delete	Audit	Audit policy record deleted	PSC-Audit
400	_sql.tbl.priv.create	Security	SQL table privilege created	PSC-Security
401	_sql.tbl.priv.update	Security	SQL table privilege updated	PSC-Security
402	_sql.tbl.priv.delete	Security	SQL table privilege deleted	PSC-Security
410	_sql.col.priv.create	Security	SQL column privilege created	PSC-Security
411	_sql.col.priv.update	Security	SQL column privilege updated	PSC-Security
412	_sql.col.priv.delete	Security	SQL column privilege deleted	PSC-Security
420	_sql.seq.priv.create	Security	SQL sequence privilege created	PSC-Security
421	_sql.seq.priv.update	Security	SQL sequence privilege updated	PSC-Security
422	_sql.seq.priv.delete	Security	SQL sequence privilege deleted	PSC-Security
500	_sys.auth.sys.create	Security	Authentication system created	PSC-Security
501	_sys.auth.sys.update	Security	Authentication system updated	PSC-Security
502	_sys.auth.sys.delete	Security	Authentication system deleted	PSC-Security
505	_sys.auth.domain.create	Security	Authentication domain created	PSC-Security

Event ID	Event name	Event type	Event description	Policy name
506	_sys.auth.domain.update	Security	Authentication domain updated	PSC-Security
507	_sys.auth.domain.delete	Security	Authentication domain deleted	PSC-Security
510	_sys.role.create	Security	Role definition created	PSC-Security
511	_sys.role.update	Security	Role definition updated	PSC-Security
512	_sys.role.delete	Security	Role definition deleted	PSC-Security
515	_sys.role.assign.create	Security	Role assignment created	PSC-Security
516	_sys.role.assign.update	Security	Role assignment updated	PSC-Security
517	_sys.role.assign.delete	Security	Role assignment deleted	PSC-Security
5000	_sys.tbl.create	Schema	Table created	App-Schema
5001	_sys.tbl.trig.create	Schema	Table trigger created	App-Schema
5002	_sys.fld.create	Schema	Table field created	App-Schema
5003	_sys.fld.trig.create	Schema	Table field trigger created	App-Schema
5004	_sys.index.create	Schema	Table index created	App-Schema
5005	_sys.index.fld.create	Schema	Table index field created	App-Schema
5010	_sys.tbl.update	Schema	Table updated	App-Schema
5111	_sys.tbl.trig.update	Schema	Table trigger updated	App-Schema
5012	_sys.fld.update	Schema	Table field updated	App-Schema
5013	_sys.fld.trig.update	Schema	Table field trigger updated	App-Schema
5014	_sys.index.update	Schema	Table index updated	App-Schema

Event ID	Event name	Event type	Event description	Policy name
5015	_sys.index.fld.update	Schema	Table index field updated	App-Schema
5020	_sys.tbl.delete	Schema	Table deleted	App-Schema
5021	_sys.tbl.trig.delete	Schema	Table trigger deleted	App-Schema
5022	_sys.fld.delete	Schema	Table field deleted	App-Schema
5023	_sys.fld.trig.delete	Schema	Table field trigger deleted	App-Schema
5024	_sys.index.delete	Schema	Table index deleted	App-Schema
5025	_sys.index.fld.delete	Schema	Table index field deleted	App-Schema
5030	_sys.seq.create	Schema	Sequence created	App-Schema
5031	_sys.seq.update	Schema	Sequence updated	App-Schema
5032	_sys.seq.delete	Schema	Sequence deleted	App-Schema
5040	_sys.db.prop.create	Schema	Database property created	App-Schema
5041	_sys.db.prop.update	Schema	Database property updated	App-Schema
5100	_sys.db.rec.create	Data	Database record created	PSC-App-Tables
5101	_sys.db.rec.update	Data	Database record updated	PSC-App-Tables
5102	_sys.db.rec.delete	Data	Database record deleted	PSC-App-Tables
10000	_sys.db.audit.enable	Audit	Database auditing enabled	PSC-Audit
10001	_sys.db.audit.disable	Audit	Database auditing disabled	PSC-Audit
10010	_sys.audit.policy.commit	Audit	Audit policy run-time update	PSC-Audit
10100	_sys.db.start	Admin	Database started	PSC-DB-Admin
10101	_sys.db.stop	Admin	Database stopped	PSC-DB-Admin

Event ID	Event name	Event type	Event description	Policy name
10202	_sys.bindump	Utility	Database binary dump	PSC-DB-Admin
10203[3]	_sys.binload	Utility	Database binary load	PSC-DB-Admin
10204	_sys.copy	Utility	Database copy	PSC-DB-Admin
10205[3]	_sys.tbl.move	Utility	Database table move	PSC-DB-Admin
10206[3]	_sys.idx.move	Utility	Database index move	PSC-DB-Admin
10207	_sys.idx.check	Utility	Database index check	PSC-DB-Admin
10208[3]	_sys.idx.rebuild	Utility	Database index rebuild	PSC-DB-Admin
10209	_sys.area.truncate	Utility	Database area truncate	PSC-DB-Admin
10211	_sys.sql.dump	Utility	Database SQL dump	PSC-DB-Admin
10212	_sys.sql.load	Utility	Database SQL load	PSC-DB-Admin
10213	_sys.txt.dump	Utility	Database text dump	PSC-DB-Admin
10214	_sys.txt.load	Utility	Database text load	PSC-DB-Admin
10215[3]	_sys.index.activation	Utility	Database index activation	PSC-DB-Admin
10216[3]	_sys.index.deactivation	Utility	Database index deactivation	PSC-DB-Admin
10217[3]	_sys.db.deallocate	Utility	Deallocate a table instance	PSC-DB-Admin
10300	_sys.audit.archive	Audit	Audit archive	PSC-Audit
10301	_sys.audit.data.dump	Audit	Audit data dump	PSC-Audit
10302	_sys.audit.data.load	Audit	Audit data load	PSC-Audit
10303	_sys.audit.pol.dump	Audit	Audit policy dump	PSC-Audit

[3] This event uses multi-tenant data formats in a multi-tenant database. For more information, see Audit data event context on page 147.

Event ID	Event name	Event type	Event description	Policy name
10304	_sys.audit.pol.load	Audit	Audit policy load	PSC-Audit
10305	_sys.audit.recoveradmin	Audit	DBA deleted last audit admin	PSC-Audit
10500	_pvm.setdbuser.success	User	Database user identity set	PSC-User
10501	_pvm.setdbuser.fail	User	Database user identity failed	PSC-User
10510	_pvm.user.login.pass	User	User login success	PSC-User
10511	_pvm.user.logout	User	User logout	PSC-User
10512	_pvm.user.login.fail	User	User login failed	PSC-User
10520	_sql.user.login.pass	User	SQL user login success	PSC-User
10521	_sql.user.logout	User	SQL user logout	PSC-User
10522	_sql.user.login.fail	User	SQL user login failed	PSC-User
10600	_pvm.db.connect	User	Database connect	PSC-User
10601	_pvm.db.disconnect	User	Database disconnect	PSC-User
10610	_sql.db.connect	User	SQL database connect	PSC-User
10611	_sql.db.disconnect	User	SQL database disconnect	PSC-User
11000	_sys.db.enc.enable	Encryption	Enable Encrypted Database feature	Database Encryption
11001	_sys.db.enc.disable	Encryption	Disable Encrypted Database feature	Database Encryption
11100	_sys.ks.create	Encryption	Create database key store	Database Encryption
11101	_sys.ks.delete	Encryption	Delete database key store	Database Encryption
11102	_sys.ks.open.pass	Encryption	Successful open database key store	Database Encryption

Event ID	Event name	Event type	Event description	Policy name
11103	_sys.ks.rekey	Encryption	Change database key store passphrase	Database Encryption
11104	_sys.ks.setcipher	Encryption	Change database key store master key type	Database Encryption
11105	_sys.ks.setadmin.pwd	Encryption	Change database key store administrator passphrase	Database Encryption
11106	_sys.ks.setuser.pwd	Encryption	Change database key store user passphrase	Database Encryption
11107	_sys.ks.ke.create.pass	Encryption	Successful create of database key store entry	Database Encryption
11108	_sys.ks.ke.update.pass	Encryption	Successful update of database key store entry	Database Encryption
11109	_sys.ks.ke.delete.pass	Encryption	Successful delete of database key store entry	Database Encryption
11110	_sys.ks.ke.read.pass	Encryption	Successful read of database key store entry	Database Encryption
11111	_sys.ks.open.fail	Encryption	Failure in database key store open	Database Encryption
11112	_sys.ks.ke.create.fail	Encryption	Failure in database key store entry create	Database Encryption
11113	_sys.ks.ke.update.fail	Encryption	Failure in database key store entry update	Database Encryption
11114	_sys.ks.ke.delete.fail	Encryption	Failure in database key store entry delete	Database Encryption
11200	_sys.as.create.pass	Encryption	Successful database auto-start file create	Database Encryption

Event ID	Event name	Event type	Event description	Policy name
11201	_sys.as.delete.pass	Encryption	Successful database auto-start file delete	Database Encryption
11202	_sys.as.open.pass	Encryption	Successful database auto-start file open	Database Encryption
11203	_sys.as.recover.pass	Encryption	Successful database auto-start file recovery	Database Encryption
11204	_sys.as.update.pass	Encryption	Successful database auto-start file update	Database Encryption
11205	_sys.as.open.fail	Encryption	Failure in database auto-start file open	Database Encryption
11206	_sys.as.update.fail	Encryption	Failure in database auto-start file update	Database Encryption
11207	_sys.as.recover.fail	Encryption	Failure in database auto-start file recovery	Database Encryption
11300	_sys.enc.scan	Encryption	Started encryption state scan of database object	Database Encryption
11301	_sys.enc.update	Encryption	Started encryption state migration of database object	Database Encryption
11400	_sys.db.dbpolicy.create	Encryption	Create new version of database security policy	Database Encryption
11401	_sys.db.dbpolicy.update	Encryption	Update database security policy version	Database Encryption
11402	_sys.db.dbpolicy.delete	Encryption	Delete database security policy version	Database Encryption
11500	_sys.db.objpolicy.create	Encryption	Create new version of database object security policy	Database Encryption

Event ID	Event name	Event type	Event description	Policy name
11501	_sys.db.objpolicy.update	Encryption	Update database object security policy version	Database Encryption
11502	_sys.db.objpolicy.delete	Encryption	Delete database object security policy version	Database Encryption
11600	_sys.db.pwdpolicy.create	Encryption	Create new database passphrase policy	Database Encryption
11601	_sys.db.pwdpolicy.update	Encryption	Update database passphrase policy	Database Encryption
11602	_sys.db.pwdpolicy.delete	Encryption	Delete database passphrase policy	Database Encryption
11700[3]	_sys.db.mt.enable	Multi-tenancy	Enable multi-tenant database feature with pass/fail info	PSC-Multi-tenancy
11701[3]	_sys.db.mt.disable	Multi-tenancy	Disable multi-tenant database feature with pass/fail info	PSC-Multi-tenancy
11710[3]	_sys.db.tenant.create	Multi-tenancy	Create tenant in multi-tenant database	PSC-Multi-tenancy
11711[3]	_sys.db.tenant.update	Multi-tenancy	Update tenant in multi-tenant database	PSC-Multi-tenancy
11712[3]	_sys.db.tenant.delete	Multi-tenancy	Delete tenant in multi-tenant database	PSC-Multi-tenancy
11713[3]	_sys.db.group.create	Multi-tenancy	Create group in multi-tenant database	PSC-Multi-tenancy
11714[3]	_sys.db.group.update	Multi-tenancy	Update group in multi-tenant database	PSC-Multi-tenancy
11715[3]	_sys.db.group.delete	Multi-tenancy	Delete group in multi-tenant database	PSC-Multi-tenancy

Event ID	Event name	Event type	Event description	Policy name
11717[3]	_sys.db.groupmember.create	Multi-tenancy	Create group member in multi-tenant database	PSC-Multi-tenancy
11718[3]	_sys.db.groupmember.update	Multi-tenancy	Update group member in multi-tenant database	PSC-Multi-tenancy
11719[3]	_sys.db.groupmember.delete	Multi-tenancy	Delete group member in multi-tenant database	PSC-Multi-tenancy
31998	`_sys.app.context`	Application	Application context	Application
31999	`_sys.app.eventgroup`	Application	Audit event group	Application
32000 and higher	user-defined	–	User-defined event	–

Index

A

AES *60*
Algorithms
  asymmetric-key *64*
  cryptographic *60*
  hashing *60*
  message digest *65*
  PBE *63*
  SSL *79*
  symmetric-key
    encryption only *65*
    encryption/decryption *60*
    modes *61*
Application
  audit-enabling *128*
  authentication and authorization *45*
  context
    auditing *99*, *121*, *129*
  defined events
    auditing *99*
  deployment security *37*
  development security *36*
  feature authorization *39*
  network security *44*
  portability *37*
  updating for auditing *131*
Application event inserter privilege *105*
Application services security *40*
AppServer
  connection security *41*
  firewall compatibility *41*
  program security *40*
Architecture of OpenEdge SSL *83*
Archiving audit data *108–109*, *115*
Assigning a database a unique identifier *116*
Asymmetric-key cryptography
  algorithms *64*
  keys *64*
  OpenEdge usage *65*
  PKI support *71*
Audit administrator
  primary *106*
  privileges *105*
  secondary *106*
Audit data archiver privilege *105*, *115*
Audit data archiving *108–109*, *115*
Audit data reporter privilege *105*
Audit event groups *121*, *149*
Audit policies
  exporting *112*
Audit policies *(continued)*
  resolving conflicts *110*
  security *109*
Audit Policy Maintenance *118–119*
Audit-event groups
  in application auditing *99*
Auditing
  security *45*
Authentication
  ABL support *45*
  application network *44*
  AppServer applications *40*
  defined *24*
  OpenEdge ABL applications *36*
  PKI support *70*
  Progress Dynamics *43*
  Report Builder *37*
  user
    external *112*, *129*
  WebClient applications *37*
Authorities for trust *72*
Authorization
  ABL support *45*
  defined *24*

B

Basic crypto *59*

C

CA *72*
CA digital certificate *73*
Cashing SSL sessions *80*
Certificate
  revocation *76*
  stores *76*
Certificate Authority *72*
Certificate store
  entries *76*
Certificate store location property *91*
Certificates, See Digital certificates
Certification Authority (CA) *72*
Certify *72*
Ciphers *60*
  See also Algorithms
Client entity *64*
client-principal object *113*
Clients
  database
    audit security *109*

Clients *(continued)*
OpenEdge
accessing SSL server X.500 Subject Name *94*
configuring and running SSL *91*
setting SSL properties and parameters *91*
SSL properties and parameters *91*
SSL *78*
Common business service *23*
Confidentiality
defined *25*
PKI support *70*
Context
auditing
database transaction *119*, *121*
Continuation records *150*
Core business service
defined *23*
Core security services *69*
Cryptographic
hash functions *65*
keys *60*
Cryptography
algorithms *60*
asymmetric-key
PKI support *71*
basic *59*
data management *66*
hashing *65*
keys *60*
message digests
PKI support *71*
OpenEdge ABL *62*
PKI support *71*
symmetric-key
PKI support *71*
Custom
audit archive tool *130*
audit policy tool *130–131*
audit reports *135*

D

Data
integrity defined *25*
integrity in a PKI *70*
privacy defined *25*
security levels in OpenEdge *38*
Data management security *38*
Data privacy
PKI support *70*
Database
audit security *109*
auditing events *98*
transaction context *121*
Database security *38*
DataServer security *39*
Decryption *59*
Default SSL identity *87*
default_server alias *87*
DES *60*
DES3 *60*
Digital certificates
life time *73*
life-cycle management *76*
revocation *76*
SSL *79*
standards *72*
storage and management *76*
X.509 public-key *73*
Digital signatures
in SSL *80*
Disabling auditing *140*
Domains
trusted registry *113*
DSA *64*

E

Enabling a database for auditing *118*
Encryption *59*, *165*
Enforce audit insert privilege option *112*
Entities
client and server *64*
Expired digital certificates
managing *76*
Exporting audit policy configuration *112*

F

Firewall compatibility
AppServer *41*
SonicMQ Adapter *42*

G

Granting audit privileges
in SQL *107*
in the 4GL *107*

H

Hash functions *65*
Hashing algorithms *60*

I

Identifying fields
event record format *147*
Integration services security *42*
Integrity
defined *25*

Integrity *(continued)*
PKI support *70*
Internal events
auditing *98*

K

Key alias
name property *89*
password property *89*
Key pair
PKI authentication *70*
public-key cryptography *64*
Key store
entries
defined *76*
OpenEdge
default alias *87*
Key store rebind *182*
Key store reconstruct *183*
Key stores
defined *76*
Keys
asymmetric *64*
cryptographic *60*
private *64*
public *64*
secret *60*
SSL *79*
symmetric *60*

L

Lifetime
digital certificate *73*
Login sessions
controlling in audit data records *113*

M

Managing cryptographic data *66*
MD5 *65*
Message Authentication Code (MAC), See Message digests
Message digests
OpenEdge usage *66*
PKI support *71*
Modes of operation
cryptographic *61*

N

Nonrepudiation
defined *26*
in auditing *104*
Nonrepudiation *(continued)*
PKI support *71*

O

Open Client application security *36*
OpenEdge ABL
application security *36*
run-time procedure authorization *39*
security
consuming Web services *41*
OpenEdge Web services *41*
source code security *37*
OpenEdge auditing
application deployment *133*, *136*
application events
application-context *99*, *129*, *148*
application-defined *99*
archiving *108*, *115*
audit event groups *99*, *121*, *149*
Audit Policy Maintenance *118–119*
context
architecture *120*
database *121*
copying database when *140*
data context *147*
data integrity *141*
data maintenance *139*
data records *113*
data recovery *140*
database backing up and restoring *139*
database events *98*
disabling *140*
enabling application for *128*
enabling database for *118*, *127*
enforce audit insert privilege *112*
evaluating requirements for
as developer *99*
as end user *100*
exporting audit policy configuration *112*
features *97*
getting started *117*
identifying fields *147*
implementation recommendations *101*
indexes
deactivating *146*
modifying *140*
internal events *98*
multi-tenant data context *147*
performance *141*
policy conflict resolution *110*
pre-10.1 databases or clients *101*
preconfigured policies *118*
privileges
application audit event inserter *105*
audit administrator *105*

OpenEdge auditing *(continued)*
privileges *(continued)*
audit data archiver *105*, *115*
audit data reporter *105*
database administrator *105*
granting in 4GL *105*, *107*
granting in SQL *105*, *107*
granting to others *105*
revoking *108*
production site configuration *134*
query guidelines *155*
querying data *142–143*
READ *129*
reports
audit event details *149*
by audit date and time *150*
client-session information *146*
custom *156*, *159*
database events *152*
event details *150*
field values *150*
internationalization *145*
orientation *156*
prefiltered *156*
prefiltered, list of *156*
run-time maintenance *141*
schema overview *144*
sealing data *110*
security
database utilities *109*
function *26*
policy *109*
separation of duty *104*
tools *109*
utilities *109*
storage management *101*, *129*, *140–141*
upgrading existing application for *136*
user ID in records *113*
using application user ID *112–113*
OpenEdge Key Store *178*
OpenEdge security
See also OpenEdge SSL
application
deployment *37*
development *36*
feature authorization *39*
network *44*
portability *37*
application services *40*
AppServer
connections *41*
firewall compatibility *41*
program *40*
asymmetric-key cryptography *65*
auditing *45*, *103*
auditing archive *108*
OpenEdge security *(continued)*
authentication and authorization *45*
client-principal object *113*
data management *38*
database *38*
DataServer *39*
integration services *42*
levels for data management *38*
Open Client applications *36*
OpenEdge ABL
applications *36*
consuming Web services *41*
OpenEdge Web services *41*
run-time procedure authorization *39*
source code *37*
overview *35*
Password-based encryption (PBE) *63*
Progress Dynamics
application options *43*
basic settings *43*
repository structures *44*
security managers *44*
Query/Results reporting *39*
Report Builder *37*
SQL data *38*
symmetric-key cryptography *62*
WebClient applications *37*
See also OpenEdge SSL
OpenEdge SSL
algorithms *79*
architecture
non-OpenEdge client and server components *85*
OpenEdge client and server components *85*
client session properties
setting *91*
clients
configuring *91*
configuring and running sessions *88*
features and services *78*
key and certificate management *80*
procedure for using *86*
server session properties
setting *89*
servers
configuring *89*
default identity *87*
establishing identity *87*
identity management *86*
trusted CA identity *88*
X.500 Subject Name *94*
session caching *80*
SSL standards supported *77*
support limitations *78*
user interactions *80*

OpenEdge Web services
security *41*

P

Password *24*
Password-based encryption (PBE)
OpenEdge usage *63*
PKI support *72*
PBE, See Password-based encryption (PBE)
Permissions *24*
PKCS#5 *63*
PKI, See Public-Key Infrastructure (PKI)
PKI trust model *72*
Preconfigured audit policies *118*
Primary audit administrator *106*
Private key
storage and management *76*
Privileges
auditing in OpenEdge
application audit event inserter *105*
audit administrator *105*
audit archiver *105*, *115*
audit data reporter *105*
database administrator *105*
granting *107*
granting to others *105*
inheriting *105*
revoking *108*
security *105*
granting in SQL *105*
granting in the 4GL *105*
PRNG *71*
Procedure for using OpenEdge SSL *86*
Progress Dynamics security
application options *43*
basic settings *43*
repository structures *44*
security managers *44*
Properties and parameters
OpenEdge SSL clients
setting *91*
OpenEdge SSL server
setting *89*
protocols, ciphers, and certificates *49*
Pseudo Random Number Generation function (PRNG) *71*
Public key *64*
Public-key cryptography
See also Asymmetric-key cryptography
PKI support *71*
See also Asymmetric-key cryptography
Public-Key Infrastructure (PKI)
core services supported
authentication *70*
confidentiality *70*
Public-Key Infrastructure (PKI) *(continued)*
core services supported *(continued)*
integrity *70*
cryptography
asymmetric-key *64*, *71*
symmetric-key *71*
password-based encryption (PBE)
PKI support *72*
support for SSL *78*
trust models and mechanisms
client and server interactions *76*
digital certificates *73*
digital signatures *75*
key and certificate management *76*

Q

Query/Results reporting security *39*
Querying audit data *143*

R

READ auditing
setting up *129*
Report Builder security *37*
Resolving audit policy conflicts *110*
Reuse a disconnected SSL session property *91*
Revoking audit privileges *108*
Root CA *72*
Root CA digital certificate *73*
Root certificate store entries *86*
RSA *64*

S

Sealing the audit data *110*
Secondary audit administrator *106*
Secret key *60*
Secret-key cryptography
PKI support *71*
Secure Sockets Layer (SSL)
features and services *78*
OpenEdge
client session properties *91*
server session properties *89*
OpenEdge limitations *78*
session components
algorithms *79*
caching *80*
identity *79*
keys and certificates *79*
standards supported *77*
trust support *80*
user interactions *80*

Security
auditing 26, 103
authentication 24
authorization 24
confidentiality 25
integrity 25
OpenEdge support 35
overview 23
trust 26
Security policies
OpenEdge ABL 62
SSL 78
Server digital certificate 73
Server entity 64
Servers
OpenEdge
configuring and running SSL 89
default SSL identity 87
managing SSL identity 86
setting SSL properties and parameters 89
SSL properties and parameters 89
trusted CA identity 88
SSL 78
Session ID
SSL 79
Session time out property 89
SET-CLIENT( ) method 113
SET-DB-CLIENT function 113
SHA-1 65
Signing
digital 75
SQL
granting auditing privileges 107
SQL application
audit-enabling 131
SQL data security 38
SSL, See Secure Sockets Layer (SSL)
SSL client 78
SSL connections enabled property 89
SSL Security
POODLE 47
SSL server 78
SSL sessions
caching 80
configuring and running OpenEdge 88
Storage management
OpenEdge audit data 101
Strength of cryptography 60
Supported
protocols, ciphers, and certificates 49
Symmetric-key cryptography
encryption-only algorithms 65
encryption/decryption algorithms 60
keys 60
OpenEdge usage 62
PKI support 71

T

Transparent Data Encryption
Autostart 180
definition 166
Disabling encryption 198
Enabling encryption 173
encryptable objects 170
Encryption policy rekey 192
Key store reconstruct 183
Keystore rebind 182
Manual start 179
OpenEdge Key Store 178
OpenEdge SQL support 195
passphrase constraints 173
Supported ciphers 173
Triple DES 60
Trust
models and relationships 26
PKI support 72
SSL support 80
Trusted CA 72
Trusted domain registry 113

U

Unique identifier (UUID)
assigning to a database 116
Upgrading
existing application for auditing 136
Use a session cache property 89
Use Application User ID for Auditing option 112
Use SSL to connect property 91
User identity
asserting in OpenEdge 104
controlling in audit records 113
User interactions
SSL 80
User login session context 121
User name 24

V

Verify the SSL server host name property 91

W

Web service consumption
security 41
WebClient security 37

X

X.500 Subject Name 94

X.509
public-key certificates
in SSL *80*

X.509 *(continued)*
standard *72*