OpenEdge Getting Started:
Core Business Services
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>Preface–1</td>
</tr>
<tr>
<td>Part I Overview</td>
<td></td>
</tr>
<tr>
<td>1. <strong>OpenEdge Core Business Services</strong></td>
<td>1–1</td>
</tr>
<tr>
<td>Security services</td>
<td>1–2</td>
</tr>
<tr>
<td>Authentication</td>
<td>1–2</td>
</tr>
<tr>
<td>Authorization</td>
<td>1–3</td>
</tr>
<tr>
<td>Auditing</td>
<td>1–3</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>1–4</td>
</tr>
<tr>
<td>Integrity</td>
<td>1–4</td>
</tr>
<tr>
<td>Trust</td>
<td>1–5</td>
</tr>
<tr>
<td>Auditing service</td>
<td>1–6</td>
</tr>
<tr>
<td>Audit trails</td>
<td>1–6</td>
</tr>
<tr>
<td>Common auditing approach</td>
<td>1–6</td>
</tr>
<tr>
<td>Audit policies</td>
<td>1–7</td>
</tr>
<tr>
<td>Audit security</td>
<td>1–8</td>
</tr>
<tr>
<td>Querying and reporting</td>
<td>1–8</td>
</tr>
<tr>
<td>Archiving audit data</td>
<td>1–9</td>
</tr>
<tr>
<td>Enabling and disabling auditing.</td>
<td>1–9</td>
</tr>
<tr>
<td>Transparent Data Encryption</td>
<td>1–10</td>
</tr>
<tr>
<td>Part II Security</td>
<td></td>
</tr>
<tr>
<td>2. <strong>Security in OpenEdge</strong></td>
<td>2–1</td>
</tr>
<tr>
<td>Application development</td>
<td>2–2</td>
</tr>
<tr>
<td>ABL applications</td>
<td>2–2</td>
</tr>
<tr>
<td>Open Client applications</td>
<td>2–3</td>
</tr>
<tr>
<td>Application deployment</td>
<td>2–4</td>
</tr>
<tr>
<td>Secure source code</td>
<td>2–4</td>
</tr>
<tr>
<td>WebClient applications</td>
<td>2–4</td>
</tr>
</tbody>
</table>
3. Cryptography ................................................................. 3–1
   Basic cryptography ......................................................... 3–2
      Algorithms ................................................................. 3–2
      Keys ................................................................. 3–2
   Symmetric (secret) key cryptography ..................... 3–3
      Encryption/decryption algorithms .............................. 3–3
      Encryption/decryption modes ........................................ 3–4
      Encryption/decryption key size ........................................ 3–4
      Combine algorithm, mode, and key size ............................. 3–5
      OpenEdge usage .......................................................... 3–6
   Password-based encryption (PBE) ................................. 3–7
      PBE algorithms ........................................................ 3–7
      OpenEdge usage ........................................................ 3–7
   Asymmetric (public) key cryptography .......................... 3–8
      Asymmetric keys ......................................................... 3–8
      Public-key algorithms .................................................. 3–9
      OpenEdge usage ........................................................ 3–9
   Message digests ........................................................ 3–10
      Message digest algorithms ............................................... 3–10
      OpenEdge usage ........................................................ 3–10
   Managing cryptographic data ........................................ 3–12
      Objectives ................................................................. 3–12
      Requirements ............................................................. 3–12
      OpenEdge solutions ...................................................... 3–12
4. **Identity Management** .................................................. 4–1
   Managing user identity .................................................. 4–2
      Authentication in OpenEdge ........................................ 4–3
   Trusted authentication systems and domains ......................... 4–5
      Trusted domain registry ........................................... 4–6
   The client-principal object ........................................... 4–8
   Asserting user IDs individually or in synchronization ............... 4–10
   Additional OpenEdge identity management options ................. 4–11
      Recording authenticated client sessions ......................... 4–11
      Disallowing the blank user ID ................................... 4–11

5. **Public-Key Infrastructure (PKI)** .................................. 5–1
   Core security services in a PKI ...................................... 5–2
      Authentication ....................................................... 5–2
      Confidentiality .................................................... 5–2
      Integrity ............................................................ 5–3
   Cryptography in a PKI ................................................ 5–4
      Symmetric-key cryptography ....................................... 5–4
      Message digests and MACs ........................................ 5–4
      Public-key cryptography .......................................... 5–5
      Password-based encryption (PBE) ................................ 5–5
   Trust relationships and supporting mechanisms ..................... 5–6
      PKI trust model .................................................... 5–6
      Digital certificates ................................................ 5–7
      Digital signatures .................................................. 5–7
      Client and server interactions ................................... 5–7
      Key and certificate management .................................. 5–8

6. **Secure Sockets Layer (SSL)** ..................................... 6–1
   SSL standards support in OpenEdge ................................ 6–2
   Features and services ................................................ 6–3
   SSL session components .............................................. 6–4
      Session identity .................................................... 6–4
      Keys and certificates .............................................. 6–4
      Algorithms (ciphers) .............................................. 6–5
      Session caching .................................................... 6–5
   Support for trust ...................................................... 6–6
   SSL interactions and the user ....................................... 6–7

7. **SSL in OpenEdge** .................................................. 7–1
   OpenEdge SSL architecture .......................................... 7–2
      OpenEdge SSL client and server components .................... 7–3
      Non-OpenEdge SSL client and server components .............. 7–4
   Using SSL in OpenEdge ............................................... 7–6
   Managing SSL server identity ...................................... 7–7
      Establishing SSL server identity ................................ 7–7
      Using the default SSL server identity ......................... 7–8
      Managing your own SSL server identity ......................... 7–8
   Configuring and running SSL sessions ............................. 7–10
      Configuring SSL server sessions ................................ 7–11
      Configuring SSL client sessions ................................ 7–13
      Accessing an SSL server’s X.500 Subject Name ................ 7–16
## Part III Auditing

### 8. Auditing in OpenEdge

- OpenEdge auditing ................................................. 8–2
  - What you can audit ............................................. 8–2
- Evaluating your auditing requirements ......................... 8–5
  - Application developer decisions ............................. 8–5
  - End user responsibilities .................................. 8–6
- OpenEdge auditing recommendations ......................... 8–7
- Auditing OpenEdge databases or clients earlier than Release 10.1 8–8
- Audit data storage management .............................. 8–9

### 9. Audit Security

- Overview ............................................................... 9–1
  - Asserting user identity ....................................... 9–2
  - Adding security through separation of duty ............... 9–2
  - Managing audit privileges .................................. 9–3
    - Inheriting audit privileges: the database administrator 9–3
    - Assigning audit security privileges .................... 9–4
    - Creating a primary and a secondary audit administrator 9–5
    - Granting audit privileges ................................ 9–6
    - Revoking audit privileges ................................. 9–7
- Audit data archival security .................................. 9–9
  - OpenEdge Audit Archiving ................................ 9–9
- Audit security for database clients, tools, and utilities .... 9–10
- Audit security for OpenEdge databases ...................... 9–11
- Audit policy security ............................................ 9–12
  - Sealing the audit data ........................................ 9–12
  - Resolving audit policy conflicts ........................... 9–13
  - Exporting audit policy configuration as an XML file .... 9–14
- Choosing additional audit options ......................... 9–15
- Configuring additional user authentication systems and domains 9–16
- Controlling the user ID and login sessions in audit data records 9–18
- Audit archiving .................................................... 9–20
  - Archiving audit data .......................................... 9–20
  - Assigning a database a unique identifier ............... 9–22

### 10. Configuring OpenEdge Auditing

- Getting started .................................................... 10–1
  - Enabling an OpenEdge database for auditing ............ 10–2
  - Connecting to the database and assigning audit-related privileges 10–4
  - Using Audit Policy Maintenance for policy creation .... 10–4
- Setting up OpenEdge auditing context ...................... 10–5
  - Auditing context architecture ............................. 10–5
  - Recording auditing context information ................. 10–7
  - How auditing context is referenced by audit event records 10–8
  - Adding auditing context to the application ............. 10–9
- Managing auditing context .................................... 10–10
- Migrating an existing application to use auditing ....... 10–11

### 11. Developing an Audit-enabled OpenEdge Application

- Developing an audit-enabled application ................. 11–1
- Audit-enabling an OpenEdge database ...................... 11–2
- Defining application audit events .......................... 11–2
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audit-enabling your OpenEdge application</td>
<td>11–3</td>
</tr>
<tr>
<td>Implementing additional auditing options</td>
<td>11–3</td>
</tr>
<tr>
<td>Updating an existing ABL application with auditing.</td>
<td>11–6</td>
</tr>
<tr>
<td>Audit-enabling your SQL application</td>
<td>11–6</td>
</tr>
<tr>
<td>12. Deploying an Audit-enabled OpenEdge Application</td>
<td>12–1</td>
</tr>
<tr>
<td>Overview</td>
<td>12–2</td>
</tr>
<tr>
<td>Developing the company’s audit policy</td>
<td>12–2</td>
</tr>
<tr>
<td>Configuring auditing at the production site</td>
<td>12–3</td>
</tr>
<tr>
<td>Applying audit policy for production systems</td>
<td>12–5</td>
</tr>
<tr>
<td>Writing custom audit reports</td>
<td>12–6</td>
</tr>
<tr>
<td>Preparing to deploy your audit-enabled application</td>
<td>12–7</td>
</tr>
<tr>
<td>Upgrading an existing application to use auditing</td>
<td>12–8</td>
</tr>
<tr>
<td>13. Maintaining Audit Data</td>
<td>13–1</td>
</tr>
<tr>
<td>Common audit data maintenance tasks</td>
<td>13–2</td>
</tr>
<tr>
<td>Backing up and restoring an audit-enabled database.</td>
<td>13–2</td>
</tr>
<tr>
<td>Archiving and loading audit tables</td>
<td>13–2</td>
</tr>
<tr>
<td>Copying an audit-enabled database</td>
<td>13–2</td>
</tr>
<tr>
<td>Recovering audit data</td>
<td>13–2</td>
</tr>
<tr>
<td>Modifying the indexes generated for OpenEdge audit tables</td>
<td>13–2</td>
</tr>
<tr>
<td>Modifying the audit table storage area</td>
<td>13–2</td>
</tr>
<tr>
<td>Disabling auditing</td>
<td>13–3</td>
</tr>
<tr>
<td>Handling long-term storage growth</td>
<td>13–3</td>
</tr>
<tr>
<td>Returning audit data from offline storage</td>
<td>13–3</td>
</tr>
<tr>
<td>Ensuring audit data integrity</td>
<td>13–3</td>
</tr>
<tr>
<td>Auditing performance</td>
<td>13–4</td>
</tr>
<tr>
<td>Run-time audit maintenance</td>
<td>13–5</td>
</tr>
<tr>
<td>Monitoring your database’s health</td>
<td>13–5</td>
</tr>
<tr>
<td>Querying audit data</td>
<td>13–5</td>
</tr>
<tr>
<td>14. Querying and Reporting on Audit Data</td>
<td>14–1</td>
</tr>
<tr>
<td>Overview</td>
<td>14–2</td>
</tr>
<tr>
<td>Audit data schema overview</td>
<td>14–2</td>
</tr>
<tr>
<td>Audit data querying and reporting</td>
<td>14–4</td>
</tr>
<tr>
<td>Reporting committed data only</td>
<td>14–4</td>
</tr>
<tr>
<td>Internationalization considerations</td>
<td>14–4</td>
</tr>
<tr>
<td>Reporting on a central archive database</td>
<td>14–4</td>
</tr>
<tr>
<td>Reporting on a single database with multiple GUIDs</td>
<td>14–5</td>
</tr>
<tr>
<td>Reporting from multiple audit databases</td>
<td>14–5</td>
</tr>
<tr>
<td>Reactivated audit indexes</td>
<td>14–5</td>
</tr>
<tr>
<td>Reporting event descriptions</td>
<td>14–5</td>
</tr>
<tr>
<td>Optional client session information</td>
<td>14–6</td>
</tr>
<tr>
<td>Audit data event context</td>
<td>14–6</td>
</tr>
<tr>
<td>Audit data application context</td>
<td>14–7</td>
</tr>
<tr>
<td>Logically grouping related audit data in an audit event group</td>
<td>14–8</td>
</tr>
<tr>
<td>Grouping audit data by transaction ID</td>
<td>14–9</td>
</tr>
<tr>
<td>Grouping audit data by database connection</td>
<td>14–9</td>
</tr>
<tr>
<td>Reporting additional audit event details</td>
<td>14–9</td>
</tr>
<tr>
<td>Reporting by audit date and time</td>
<td>14–9</td>
</tr>
<tr>
<td>Reporting by user ID</td>
<td>14–9</td>
</tr>
<tr>
<td>Reporting modified field old/new values</td>
<td>14–10</td>
</tr>
<tr>
<td>Identifying database events</td>
<td>14–12</td>
</tr>
<tr>
<td>Internal audit event policies</td>
<td>14–13</td>
</tr>
<tr>
<td>Efficient index use</td>
<td>14–13</td>
</tr>
<tr>
<td>Query guidelines</td>
<td>14–16</td>
</tr>
</tbody>
</table>
Reporting on audit data with prefiltered and custom reports .................................. 14–17
Generating a custom filtered report ................................................................. 14–21

Part IV  Transparent Data Encryption

15.  Transparent Data Encryption in OpenEdge RDBMS ........................................... 15–1
    Defining the problem space ................................................................. 15–2
      Providing data privacy ................................................................. 15–2
    Defining Transparent Data Encryption ................................................. 15–3
    Data security .................................................................................. 15–4
      Data Storage ............................................................................. 15–4
      Encrypted Data Configuration ...................................................... 15–5
      Encryption Key ........................................................................... 15–5
    Before you start .............................................................................. 15–7
    What is encryptable ......................................................................... 15–9
      Data considerations ..................................................................... 15–9
    Transparent Data Encryption feature summary .................................. 15–10
    Enabling Encryption ......................................................................... 15–11
      Adding an encryption area ............................................................. 15–11
      Executing the ENABLEENCRYPTION command ......................... 15–12

16.  OpenEdge Key Store ................................................................................. 16–1
    General key store basics ..................................................................... 16–2
    OpenEdge key store .......................................................................... 16–4
    Configuring key store access ............................................................. 16–5
      Manual start ................................................................................. 16–5
      Autostart ....................................................................................... 16–6
    Key store maintenance ..................................................................... 16–8
      Backups ......................................................................................... 16–8
      Modifying passphrases. ................................................................. 16–8
      Rebind ........................................................................................... 16–9
      Key store reconstruct ................................................................... 16–10

17.  Configuring Transparent Data Encryption policies ........................................ 17–1
    Creating encryption policies ............................................................... 17–2
      Creating encryption policies with PROUTIL EPOLICY. .................. 17–2
      Creating encryption policies with Data Admin ............................... 17–3
    Encrypting your existing data ............................................................. 17–6
      EPOLICY MANAGE UPDATE EXAMPLE ...................................... 17–6
      All new data is encrypted ............................................................... 17–8
    Maintaining Transparent Data Encryption .......................................... 17–9
      Rekey encryption policies with Data Admin ..................................... 17–10
      Rekey an encryption policy with PROUTIL EPOLICY .................... 17–11
      Change the cipher of an encryption policy with PROUTIL EPOLICY. 17–12
      Viewing encryption policy history in Data Admin ........................... 17–12
    OpenEdge SQL support for Transparent Data Encryption ..................... 17–13
      Using the CREATE TABLE statement .............................................. 17–13
      Adding an encrypted column ......................................................... 17–14
      Using the CREATE INDEX statement ............................................. 17–14
      Using the ALTER TABLE statement ............................................. 17–15
      Viewing encryption policy with the SHOW ENCRYPT statement .... 17–16
      Deleting encrypted objects with OpenEdge SQL ......................... 17–16
    Disabling encryption ......................................................................... 17–17
Part V Appendixes

A. **Audit Data Tables** ................................................................. A–1
   Audit data table schema ......................................................... A–2

B. **Preconfigured Audit Policies** ................................................ B–1
   Using the preconfigured OpenEdge audit policies .................... B–2

Index ................................................................. Index–1
Figures

Figure 1–1: Security layers in an OpenEdge application ........................................ 1–10
Figure 3–1: Relative key strength ........................................................................... 3–5
Figure 3–2: Relative key encryption and decryption speed ..................................... 3–5
Figure 7–1: SSL architecture in OpenEdge (See Figure 7–2 for key to diagram.) .... 7–2
Figure 7–2: Key to OpenEdge SSL architecture in Figure 7–1 ............................... 7–3
Figure 10–1: Auditing context levels ...................................................................... 10–6
Figure 10–2: Auditing record relationship .............................................................. 10–9
Figure 14–1: Audit tables schema ......................................................................... 14–2
Figure 14–2: Detail orientation report output ......................................................... 14–17
Figure 14–3: Summary orientation report output ................................................... 14–18
Figure 14–4: Sample custom audit data filter report .............................................. 14–21
Figure 14–5: Custom audit data filter report detail .............................................. 14–22
Figure 17–1: Encrypted data life cycle .................................................................... 17–9
Figure A–1: OpenEdge auditing metadata .............................................................. A–2
Tables

Table 4–1: OpenEdge user identities ....................................................... 4–2
Table 7–1: OpenEdge server SSL session properties .............................. 7–11
Table 7–2: Mechanisms for setting OpenEdge SSL server session properties .... 7–12
Table 7–3: OpenEdge client SSL session properties .............................. 7–13
Table 7–4: Mechanisms for setting OpenEdge SSL client session properties .... 7–14
Table 7–5: Mechanisms for accessing the SSL server X.500 Subject name .... 7–16
Table 9–1: Granting audit privileges to other users ............................... 9–5
Table 14–1: Sample data type values ................................................. 14–11
Table 14–2: _aud-audit-data main table indexed fields ............................ 14–14
Table 14–3: _aud-audit-data value child table indexed fields ..................... 14–15
Table 14–4: Prefiltered audit data reports .......................................... 14–18
Table 15–1: Database Master key ciphers ......................................... 15–13
Table 15–2: Passphrase constraints ............................................. 15–14
Table 17–1: Object ciphers .......................................................... 17–2
Table A–1: OpenEdge audit tables .................................................. A–3
Table A–2: _aud-audit-data table .................................................... A–8
Table A–3: _aud-audit-data-value table ............................................... A–14
Table A–4: _aud-audit-policy table ................................................... A–17
Table A–5: _aud-event table .......................................................... A–19
Table A–6: _aud-event-policy table ................................................ A–20
Table A–7: _aud-field-policy table .................................................. A–21
Table A–8: _aud-file-policy table ...................................................... A–26
Table A–9: _client-session table ..................................................... A–31
Table A–10: _db table ................................................................. A–33
Table A–11: _db-detail table .......................................................... A–34
Table A–12: _db-option table .......................................................... A–36
Table B–1: Predefined OpenEdge database audit policy events ............. B–2
Preface

This Preface contains the following sections:

- Purpose
- Audience
- Organization
- Using this manual
- Typographical conventions
- Examples of syntax descriptions
- OpenEdge messages
- Third party acknowledgements
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

Part I, Overview

Chapter 1, “OpenEdge Core Business Services”

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

Part II, Security

Chapter 2, “Security in OpenEdge”

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

Chapter 3, “Cryptography”

Defines cryptography and describes how it is supported in OpenEdge.

Chapter 4, “Identity Management”

Describes authentication and authorization methods supported by OpenEdge, including using the OpenEdge database _User table or an external user identity validation source.
Chapter 5, “Public-Key Infrastructure (PKI)”

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

Chapter 6, “Secure Sockets Layer (SSL)”

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

Chapter 7, “SSL in OpenEdge”

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

Part III, Auditing

Chapter 8, “Auditing in OpenEdge”

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

Chapter 9, “Audit Security”

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.

Chapter 10, “Configuring OpenEdge Auditing”

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

Chapter 11, “Developing an Audit-enabled OpenEdge Application”

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

Chapter 12, “Deploying an Audit-enabled OpenEdge Application”

Provides details about audit-enabled OpenEdge application deployment.

Chapter 13, “Maintaining Audit Data”

Describes common audit data record maintenance tasks.

Chapter 14, “Querying and Reporting on Audit Data”

Describes how to query and report on data in the audit tables.
Part IV, Transparent Data Encryption

Chapter 15, “Transparent Data Encryption in OpenEdge RDBMS”

Defines and describes Transparent Data Encryption in the OpenEdge RDBMS.

Chapter 16, “OpenEdge Key Store”

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

Chapter 17, “Configuring Transparent Data Encryption policies”

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

Part V, Appendixes

Appendix A, “Audit Data Tables”

Describes all the audit data tables.

Appendix B, “Preconfigured Audit Policies”

Describes each of the preconfigured audit policies included with OpenEdge.

Using this manual

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 see the OpenEdge Product Documentation Overview page on PSDN: http://communities.progress.com/pcom/docs/DOC-16074.

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.

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
## Typographical conventions

This manual uses the following typographical conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.</td>
</tr>
<tr>
<td><strong>Italic</strong></td>
<td>Italic typeface indicates the title of a document, or signifies new terms.</td>
</tr>
<tr>
<td><strong>SMALL, BOLD CAPITAL LETTERS</strong></td>
<td>Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, GET and CTRL.</td>
</tr>
<tr>
<td><strong>KEY1+KEY2</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>KEY1 KEY2</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

**Syntax:**

- **Fixed width** A fixed-width font is used in syntax statements, code examples, system output, and filenames.
- **Fixed-width italics** Fixed-width italics indicate variables in syntax statements.
- **Fixed-width bold** Fixed-width bold indicates variables with special emphasis.
- **UPPERCASE fixed width** Uppercase words are ABL keywords. Although these are always shown in uppercase, you can type them in either uppercase or lowercase in a procedure.

- ![icon](image) This icon (three arrows) introduces a multi-step procedure.
- ![icon](image) This icon (one arrow) introduces a single-step procedure.

- **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.
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

\[
\text{MAXIMUM (expression , expression [ , expression ] ...)}
\]

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

\[
\text{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

\[
\{\text{include-file}
  \hspace{1em}\begin{aligned} &\text{argument |} \\
  &\hspace{1em}\&\text{argument-name = "argument-value"} \end{aligned}\}
\]

Long syntax descriptions split across lines

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

In this example, WITH is followed by six optional items:

Syntax

\[
\text{WITH \begin{aligned} &\text{ACCUM max-length} \end{aligned} \begin{aligned} &\text{expression DOWN} \end{aligned} \\
  \hspace{1em}\begin{aligned} &\text{CENTERED} \end{aligned} \begin{aligned} &\text{n COLUMNS} \end{aligned} \begin{aligned} &\text{SIDE-LABELS} \end{aligned} \\
  \hspace{1em}\begin{aligned} &\text{STREAM-IO} \end{aligned} \]
\]

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**

\[
\text{ASSIGN} \quad \{ \begin{array}{l}
\quad \{ \text{FRAME} \text{ frame} \} \{ \text{field} \ [ = \text{expression} ] \} \\
\quad \{ \text{WHEN} \text{ expression} \} \ldots \\
\quad | \{ \text{record} \ [ \text{EXCEPT} \text{ field} \ldots \] \}
\end{array} \}
\]

**OpenEdge messages**

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

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.

- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.

- **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

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

- Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.

- Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.

- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.

- Terminates the current session.

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

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

If you encounter an error that terminates OpenEdge, note the message number before restarting.
Obtaining more information about OpenEdge messages

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

- Choose Help → Recent Messages to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.
- Choose Help → Messages and then type the message number to display a description of a specific OpenEdge message.
- In the Procedure Editor, press the HELP key or F1.

On UNIX platforms, use the OpenEdge pro command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

To use the pro command to obtain a message description by message number:

1. Start the Procedure Editor:

   OpenEdge-install-dir/bin/pro

2. Press F3 to access the menu bar, then choose Help → Messages.
3. Type the message number and press ENTER. Details about that message number appear.
4. Press F4 to close the message, press F3 to access the Procedure Editor menu, and choose File → Exit.

Third party acknowledgements

OpenEdge includes AdventNet - Agent Toolkit licensed from AdventNet, Inc. http://www.adventnet.com. All rights to such copyright material rest with AdventNet.

OpenEdge includes ANTLR (Another Tool for Language Recognition) software Copyright © 2003-2006, Terence Parr All rights reserved. Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes software developed by the Apache Software Foundation (http://www.apache.org/). Copyright © 1999 The Apache Software Foundation. All rights reserved (Xerces C++ Parser (XML) and Xerces2 Java Parser (XML)); Copyright © 1999-2002 The Apache Software Foundation. All rights reserved (Xerces Parser (XML); and Copyright © 2000-2003 The Apache Software Foundation. All rights reserved (Ant). The names “Apache,” “Xerces,” “ANT,” and “Apache Software Foundation” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “Apache”, nor may “Apache” appear in their name, without
prior written permission of the Apache Software Foundation. For written permission, please contact apache@apache.org. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes Concurrent Java software Copyright 1994-2000 Sun Microsystems, Inc. All Rights Reserved. Neither the name of or trademarks of Sun may be used to endorse or promote products including or derived from the Java Software technology without specific prior written permission; and Redistributions of source or binary code must contain the above copyright notice, this notice and the following disclaimers: This software is provided “AS IS,” without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN MICROSYSTEMS, INC. OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN MICROSYSTEMS, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

OpenEdge includes DataDirect software Copyright © 1991-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for JDBC Type 4 driver); Copyright © 1993-2009 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for JDBC); Copyright © 1988-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for ODBC); and Copyright © 1988-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect64 for ODBC).

OpenEdge includes DataDirect Connect for ODBC and DataDirect Connect64 for ODBC software, which include ICU software 1.8 and later - Copyright © 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

OpenEdge includes DataDirect Connect for ODBC and DataDirect Connect64 for ODBC software, which include software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Copyright © 1998-2006 The OpenSSL Project. All rights reserved. And Copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved.

OpenEdge includes DataDirect products for the Microsoft SQL Server database which contain a licensed implementation of the Microsoft TDS Protocol.

OpenEdge includes software authored by David M. Gay. Copyright © 1991, 2000, 2001 by Lucent Technologies (dtoa.c); Copyright © 1991, 1996 by Lucent Technologies (g_fmt.c); and
Copyright © 1991 by Lucent Technologies (rnd_prod.s). Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which includes or accompanies a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

OpenEdge includes software authored by David M. Gay. Copyright © 1998-2001 by Lucent Technologies All Rights Reserved (decsrtod.c; strtodg.c); Copyright © 1998, 2000 by Lucent Technologies All Rights Reserved (decsstrtof.c; strtord.c); Copyright © 1998 by Lucent Technologies All Rights Reserved (dmisc.c; gdtoa.h; gethex.c; gmisc.c; sum.c); Copyright © 1998, 1999 by Lucent Technologies All Rights Reserved (gdtoa.c; misc.c; smisc.c; ulp.c); Copyright © 1998-2000 by Lucent Technologies All Rights Reserved (gdtoaimp.h); Copyright © 2000 by Lucent Technologies All Rights Reserved (hd_init.c). Full copies of these licenses can be found in the installation directory, in the c:/OpenEdge/licenses folder. Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

OpenEdge includes http package software developed by the World Wide Web Consortium. Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All rights reserved. This work is distributed under the W3C® Software License [http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231] in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

OpenEdge includes ICU software 1.8 and later - Copyright © 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.


OpenEdge includes Infragistics NetAdvantage for .NET v2009 Vol 2 Copyright © 1996-2009 Infragistics, Inc. All rights reserved.
OpenEdge includes JSTL software Copyright 1994-2006 Sun Microsystems, Inc. All Rights Reserved. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes OpenSSL software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Copyright © 1998-2007 The OpenSSL Project. All rights reserved. This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com). Copyright © 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project. Software distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes Quartz Enterprise Job Scheduler software Copyright © 2001-2003 James House. All rights reserved. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product. This product uses and includes within its distribution, software developed by the Apache Software Foundation (http://www.apache.org/).

OpenEdge includes code licensed from RSA Security, Inc. Some portions licensed from IBM are available at http://oss.software.ibm.com/icu4j/.

OpenEdge includes the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright ©1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

OpenEdge includes Sonic software, which includes software developed by Apache Software Foundation (http://www.apache.org/). Copyright © 1999-2000 The Apache Software Foundation. All rights reserved. The names “Ant”, “Axis”, “Xalan,” “FOP,” “The Jakarta Project”, “Tomcat”, “Xerces” and/or “Apache Software Foundation” must not be used to endorse or promote products derived from the Product without prior written permission. Any product derived from the Product may not be called “Apache”, nor may “Apache” appear in their name, without prior written permission. For written permission, please contact apache@apache.org.

OpenEdge includes Sonic software, which includes software Copyright © 1999 CERN - European Organization for Nuclear Research. Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. CERN makes no representations about the suitability of this software for any purpose. It is provided "as is" without expressed or implied warranty.

OpenEdge includes Sonic software, which includes software developed by ExoLab Project (http://www.exolab.org/). Copyright © 2000 Intalio Inc. All rights reserved. The names “Castor” and/or “ExoLab” must not be used to endorse or promote products derived from the Products without prior written permission. For written permission, please contact info@exolab.org. Exolab, Castor and Intalio are trademarks of Intalio Inc.
OpenEdge includes Sonic software, which includes software developed by IBM. Copyright © 1995-2003 International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. Software distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

OpenEdge includes Sonic software, which includes the JMX Technology from Sun Microsystems, Inc. Use and Distribution is subject to the Sun Community Source License available at http://sun.com/software/communitysource.

OpenEdge includes Sonic software, which includes software developed by the ModelObjects Group (http://www.modelobjects.com). Copyright © 2000-2001 ModelObjects Group. All rights reserved. The name “ModelObjects” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “ModelObjects”, nor may “ModelObjects” appear in their name, without prior written permission. For written permission, please contact djacobs@modelobjects.com.

OpenEdge includes Sonic software, which includes code licensed from Mort Bay Consulting Pty. Ltd. The Jetty Package is Copyright © 1998 Mort Bay Consulting Pty. Ltd. (Australia) and others.

OpenEdge includes Sonic software, which includes files that are subject to the Netscape Public License Version 1.1 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/NPL/. Software distributed under the License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is Mozilla Communicator client code, released March 31, 1998. The Initial Developer of the Original Code is Netscape Communications Corporation. Portions created by Netscape are Copyright 1998-1999 Netscape Communications Corporation. All Rights Reserved.

OpenEdge includes Sonic software, which includes software developed by the University Corporation for Advanced Internet Development http://www.ucaid.edu Internet2 Project. Copyright © 2002 University Corporation for Advanced Internet Development, Inc. All rights reserved. Neither the name of OpenSAML nor the names of its contributors, nor Internet2, nor the University Corporation for Advanced Internet Development, Inc., nor UCAID may be used to endorse or promote products derived from this software and products derived from this software may not be called OpenSAML, Internet2, UCAID, or the University Corporation for Advanced Internet Development, nor may OpenSAML appear in their name without prior written permission of the University Corporation for Advanced Internet Development. For written permission, please contact opensaml@opensaml.org.

OpenEdge includes the UnixWare platform of Perl Runtime authored by Kiem-Phong Vo and David Korn. Copyright © 1991, 1996 by AT&T Labs. Permission to use, copy, modify, and
Preface

distribute this software for any purpose without fee is hereby granted, provided that this entire
notice is included in all copies of any software which is or includes a copy or modification of
this software and in all copies of the supporting documentation for such software. THIS
SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED
WARRANTY. IN PARTICULAR, NEITHER THE AUTHORS NOR AT&T LABS MAKE
ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE
MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR
PURPOSE.

OpenEdge includes Vermont Views Terminal Handling Package software developed by

OpenEdge includes XML Tools, which includes versions 8.9 of the Saxon XSLT and XQuery
Processor from Saxonica Limited (http://www.saxonica.com/) which are available from
SourceForge (http://sourceforge.net/projects/saxon/). The Original Code of Saxon
comprises all those components which are not explicitly attributed to other parties. The Initial
Developer of the Original Code is Michael Kay. Until February 2001 Michael Kay was an
employee of International Computers Limited (now part of Fujitsu Limited), and original code
developed during that time was released under this license by permission from International
Computers Limited. From February 2001 until February 2004 Michael Kay was an employee
of Software AG, and code developed during that time was released under this license by
permission from Software AG, acting as a "Contributor". Subsequent code has been developed
by Saxonica Limited, of which Michael Kay is a Director, again acting as a "Contributor". A
small number of modules, or enhancements to modules, have been developed by other
individuals (either written especially for Saxon, or incorporated into Saxon having initially been
released as part of another open source product). Such contributions are acknowledged
individually in comments attached to the relevant code modules. All Rights Reserved. The
contents of the Saxon files are subject to the Mozilla Public License Version 1.0 (the "License");
you may not use these files except in compliance with the License. You may obtain a copy of
the License at http://www.mozilla.org/MPL/ and a copy of the license can also be found in the
installation directory, in the c:/OpenEdge/licenses folder. Software distributed under the
License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either
express or implied. See the License for the specific language governing rights and limitations
under the License.

OpenEdge includes XML Tools, which includes Xs3P v1.1.3. The contents of this file are
subject to the DSTC Public License (DPL) Version 1.1 (the "License"); you may not use this
file except in compliance with the License. A copy of the license can be found in the installation
directory, in the c:/OpenEdge/licenses folder. Software distributed under the License is
distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or
implied. See the License for the specific language governing rights and limitations under the
License. The Original Code is xs3p. The Initial Developer of the Original Code is DSTC.
Portions created by DSTC are Copyright © 2001, 2002 DSTC Pty Ltd. All rights reserved.

OpenEdge includes YAJL software Copyright 2007, Lloyd Hilaiel. Redistribution and use in
source and binary forms, with or without modification, are permitted provided that the
following conditions are met: 1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form
must reproduce the above copyright notice, this list of conditions and the following disclaimer
in the documentation and/or other materials provided with the distribution. 3. Neither the name
of Lloyd Hilaiel nor the names of its contributors may be used to endorse or promote products
derived from this software without specific prior written permission. THIS SOFTWARE IS
PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Part I

Overview

Chapter 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.

The following sections provide an overview of the core business services currently supported by OpenEdge:

- Security services
- Auditing service
- Transparent Data Encryption
Security services

Security is a service that consists of several core services that support the following basic security functions:

- Authentication
- Authorization
- Auditing

Authentication, authorization, and auditing implement the following attributes:

- Confidentiality
- Integrity
- Trust

The following sections describe these basic security functions and attributes and how they are supported in OpenEdge. For more information on particular OpenEdge components and the specific security features that they support, see Chapter 2, “Security in OpenEdge.”

Authentication

Authentication assures that the identity asserted by one entity can be validated as authentic. For example, when a user logs on to an application, the authentication service assures that the user is permitted to access the application and its features. This concept of an identity can be extended from a user to various components of an application distributed across a network that need to communicate securely with one another.

The minimum requirement to authenticate a user is typically some name that uniquely identifies the user (user ID or user name) and a secret key (password) that the user provides to validate the user’s name. If the authentication service can associate the password with the user name, it establishes the user name as a valid user of the application.

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 the connection between these components. 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 the “Auditing” section on page 1–3).

In OpenEdge, the authentication of user names is supported through various user identity management features. These features include a user authentication mechanism for use with the OpenEdge RDBMS, which can maintain its own list of valid users. The features also include a mechanism to assert a previously authenticated user ID as the current application/database connection user. For more information on OpenEdge support for authentication, see Chapter 2, “Security in OpenEdge.”
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** — See Chapter 5, “Public-Key Infrastructure (PKI)”
- **SSL** — See Chapter 6, “Secure Sockets Layer (SSL)”
- **Support for SSL in OpenEdge** — See Chapter 7, “SSL in OpenEdge”

**Authorization**

*Authorization* grants or denies an entity access to capabilities based on the entity’s validated identity. Thus, an authorization service acts, based on the results of the authentication service, to authorize a user access to restricted application features and data. Two common authorization mechanisms are *permissions* and *user roles*. Permissions associate users with capabilities. Once authenticated, a given user automatically inherits the associated capabilities. User roles are predefined (or assignable) user categories that permit access to well-defined capabilities. For example, a security administrator role might allow any user who is a member of that role to manage user accounts and all other security functions in an application.

In OpenEdge, authorization is supported using permissions in ABL (Advanced Business Language) for AVM compile-time features and run-time functions to match users against capabilities defined in the OpenEdge RDBMS. OpenEdge supports authorization based on user roles for managing the Web Services Adapter (WSA). For more information on OpenEdge support for authorization, see Chapter 2, “Security in OpenEdge.”

**Auditing**

*Auditing* is the action of using a trail of secure and nonrepudiatable events that transpired during the execution of a business process in order to detect when that process has not been followed or changed, or the process-data 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 unauthorized user (determined by other means) manages to successfully assert another user’s identity, auditing will record the exact time and means of access through the stolen identity. Further, 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. For more information on auditing in OpenEdge, see the “Auditing service” section on page 1–6.

**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 beforehand. 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 infrastructure to connect the distributed components of a deployed Internet or intranet application. For more information on:

- **OpenEdge support for cryptography** — See Chapter 3, “Cryptography”
- **OpenEdge support for data privacy in a distributed application** — See Chapter 5, “Public-Key Infrastructure (PKI)”

**Integrity**

*Integrity* assures that data has not been altered (intentionally or unintentionally) 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 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 Chapter 3, “Cryptography.”
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.

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** — See Chapter 5, “Public-Key Infrastructure (PKI)”
- **The elements of PKI implemented by OpenEdge** — See Chapter 5, “Public-Key Infrastructure (PKI)” and Chapter 6, “Secure Sockets Layer (SSL)”
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 the “Getting started” section on page 10–2 and the Audit Policy Maintenance Help.

The OpenEdge auditing solution includes:

- Audit trails
- Common auditing approach
- Audit policies
- Audit security
- Querying and reporting
- Archiving audit data
- Enabling and disabling auditing

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 policies 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 Appendix B, “Preconfigured Audit Policies.”

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 the “Resolving audit policy conflicts” section on page 9–13.

- **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

OpenEdge audit security allows you to use an external authentication system to validate user identity. You can use this capability with auditing to establish an external system, such as an ABL application, as an authentication source trusted by OpenEdge. Although you can continue to use the validation of user name and password in the _User table, you can configure your application to use its own authentication system if you prefer.

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 Chapter 9, “Audit Security.”

Querying and reporting

Because the audit data is stored in standard OpenEdge database tables, you can easily query against the table’s 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 the audit data, choosing to generate 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 Appendix A, “Audit Data Tables.” For more information about querying and reporting, see Chapter 14, “Querying and Reporting on Audit Data.”
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 Chapter 9, “Audit Security.”

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 Figure 1–1.

Figure 1–1: 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 data’s 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 company’s network security or obtain a copy of your database through other methods. Possible external intruders include a person who:

- Has accessed a company’s 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 username 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 Part IV, “Transparent Data Encryption.” The chapters include:
  – Chapter 15, “Transparent Data Encryption in OpenEdge RDBMS”
  – Chapter 16, “OpenEdge Key Store”
  – Chapter 17, “Configuring Transparent Data Encryption policies”

• 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.
Part II

Security

Chapter 2, Security in OpenEdge
Chapter 3, Cryptography
Chapter 4, Identity Management
Chapter 5, Public-Key Infrastructure (PKI)
Chapter 6, Secure Sockets Layer (SSL)
Chapter 7, SSL in OpenEdge
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, as outlined in the following sections:

- Application development
- Application deployment
- Data management
- Application feature authorization
- Application service security features
- Integration service security features
- WebSpeed security features
- Progress Dynamics security features
- Application network security using SSL
- Auditing security
- Core user authentication and authorization
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:

- General user authentication through user ID and password validation to support other security features. For more information on user authentication features, see the “Core user authentication and authorization” section on page 2–18.
- Database connection authorization to ensure authorized access to a database.
- Data privacy using the Secure Sockets Layer (SSL) between ABL client socket objects and supported socket servers, and between supported socket clients and ABL server socket objects; also between the ABL client and database, and between the ABL client and AppServer. For more information on SSL in OpenEdge, see the “Application network security using SSL” section on page 2–16.
- Schema authorization to ensure that only authorized users can modify table, field, and index definitions.
- Compile-time authorization to ensure that only authorized users can compile procedures that perform specific database table and field accesses.
- 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.
- Cryptography to ensure that unauthorized users cannot read or change data. For more information on cryptography and the cryptographic features of ABL, see Chapter 3, “Cryptography.”

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.

For more information on authorization, see the “Core user authentication and authorization” section on page 2–18. 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 the “Application network security using SSL” section on page 2–16.

- Support Java Open Clients using digitally signed Java applets. For more information, see OpenEdge Development: Java 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
- WebClient applications
- Report Builder user authentication
- Application portability

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 the “Application development” section on page 2–2). 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 (see the “Database security” section on page 2–6).
- Schema authorization (see the “Database security” section on page 2–6).
- Compile-time authorization, which requires ABL procedures to have authorization to compile with a particular database configured using a DBAUTHKEY configuration option. For information on compile-time authorization, see *OpenEdge Deployment: Managing ABL Applications* and on DBAUTHKEY, see *OpenEdge Data Management: Database Administration*.
- Run-time authorization, which includes the option to restrict the type of server (SQL or ABL) that accesses the OpenEdge RDBMS. For information on run-time authorization, see *OpenEdge Deployment: Managing ABL Applications* 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 Chapter 3, “Cryptography.”

**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.

For more information on authorization in OpenEdge, see the “Core user authentication and authorization” section on page 2–18.
**Database security**

The OpenEdge RDBMS provides the following security features:

- 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*.

- Creating embedded lists of users who are authenticated to use the database.

- Assigning a security administrator, setting up users, 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).

- Setting up schema authorization.

- Sealing auditing data using a MAC key. For more information on MAC keys, see Chapter 3, “Cryptography.” For more information on using MAC keys to seal auditing data, see Part III, “Auditing.”

- 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 the “Application network security using SSL” section on page 2–16.

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

**DataServer security**

For DataServers, OpenEdge provides guidelines to balance OpenEdge and data source security requirements. For more information, see the appropriate DataServer manual:

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

- *OpenEdge Data Management: DataServer for ODBC*

- *OpenEdge Data Management: DataServer for Oracle*

---

**Note:** OpenEdge does not support SSL client connections to DataServers.
OpenEdge SQL data security

OpenEdge support for SQL data security supports some standard OpenEdge data security features as well as some features unique to SQL, or that otherwise distinguish SQL from ABL data security, such as:

- The relationship between OpenEdge SQL security, ABL security, and the database, including the option to restrict access to the database to either SQL or ABL using the Type of Server to Start (-ServerType) database startup parameter. For information on the -ServerType startup parameter, see OpenEdge Deployment: Startup Command and Parameter Reference.

- Authentication and creating users and administrators.

- Authorization through the granting, verifying, and revoking of privileges.

- Data privacy over connections between SQL clients and the database using SSL. For more information on SSL in OpenEdge, see the “Application network security using SSL” section on page 2–16.

For more information on authorization and authentication in OpenEdge SQL, see the “Core user authentication and authorization” section on page 2–18. For more information on SQL data security, see OpenEdge Data Management: SQL Development.
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 the “ABL applications” section on page 2–2 and the “Core user authentication and authorization” section on page 2–18.

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 the “Progress Dynamics security features” section on page 2–13.
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 the following features for securing an application running on the OpenEdge AppServer:

- Authenticating users to the AppServer at connect time.
- 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 API 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 the “Secure source code” section on page 2–4). You might also want to implement secure AppServer application services using the completely dynamic application environment provided by Progress Dynamics (see the “Progress Dynamics security features” section on page 2–13).

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) 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*.

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 the “Application network security using SSL” section on page 2–16.

**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 the “Application network security using SSL” section on page 2–16. For more information on OpenEdge integration services and their architectures, see OpenEdge Getting Started: Application and Integration Services.
WebSpeed security features

WebSpeed® is an application service technology for Web browser applications. OpenEdge supports a number of special-purpose security options for WebSpeed that are tailored for the WebSpeed environment, including:

- Options for different working modes, including Evaluation, Development, and Production that provide different levels of access to the WebSpeed WebTools.

- Configuration options to minimize security risks, including options for:
  - Special port numbers and WebSpeed server names
  - Minimizing the number of PROPATH entries
  - Minimizing unauthorized access to WebSpeed Messengers
  - Restricting file upload directories
  - Additional WebSpeed secure configuration considerations

- Password authentication using SpeedScript®.

- Data privacy over the connection between the WebSpeed Messenger and the WebSpeed Transaction Server using SSL. For more information on SSL in OpenEdge, see the “Application network security using SSL” section on page 2–16.

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

- Maximizing compatibility with firewalls, including:
  - TCP/IP port management
  - Network Address Translation (NAT) and distributed configurations
  - NameServer client port range value settings
  - Direct connection to the WebSpeed® Transaction Server without a NameServer to avoid UDP conflicts with firewalls

For more information on WebSpeed security options, see OpenEdge Getting Started: WebSpeed Essentials. For more information on WebSpeed and its architecture, 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 ABL’s 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
- Defining Progress Dynamics application security options
- Using the Progress Dynamics security managers
- Understanding the Repository security structures

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)
Security in OpenEdge

- 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 Chapter 5, “Public-Key Infrastructure (PKI)”

- The features and functions of SSL, see Chapter 5, “Public-Key Infrastructure (PKI)”

- How OpenEdge uses SSL to secure its client, server, and middleware components, and how to configure these components to use SSL, see Chapter 6, “Secure Sockets Layer (SSL)”

- 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 Chapter 1, “OpenEdge Core Business Services”) 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 the “Core user authentication and authorization” section on page 2–18.

- **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 the “Core user authentication and authorization” section on page 2–18.

- **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:

- Creating user IDs in the OpenEdge RDBMS _User table
- Authenticating and authorizing a user ID to the OpenEdge RDBMS _User table
- Asserting and authorizing access for a user ID validated from a trusted authentication system

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 ID to the OpenEdge RDBMS _User table

In ABL, you can authenticate a user ID found in the OpenEdge RDBMS _User table using the following features:

- **User ID (-U) and Password (-P) startup parameters** — Authenticates the user ID specified on the command line or in the CONNECT statement to the user IDs and passwords stored in the OpenEdge RDBMS.
- **SETUSERID function** — Authenticates a specified user ID and password to the user IDs and passwords stored in the OpenEdge RDBMS. If authenticated, this function also asserts the user ID as the current user ID, as returned by the USERID function.
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-DO function** — Use this function to authorize one or more user IDs against a list of permissions that specify whether a user can access certain procedures or features.

- **Can-* permissions** — Specify these permissions through OpenEdge Data Administration to authorize the connected user ID for table and field operations at ABL compile time and (optionally) at run time.

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

**Asserting and authorizing access for a user ID validated from a trusted authentication system**

In ABL, you can use the SETUSERID function to assert a current user for a user ID authenticated and stored in the OpenEdge RDBMS. However, to assert a user that is authenticated by any external trusted authentication system (independent of _User table user IDs), ABL and OpenEdge RDBMS provide the following elements:

- Client-principal object
- SECURITY–POLICY system handle
- SET–DB–CLIENT function
- OpenEdge RDBMS authentication system and domain tables

Together, these ABL mechanisms allow you to assert a user from the specified authentication domain as the connected application or database user. This connected user identity can be stored and passed around an application independent of the database. For more information on managing identities for both application and database access, see Chapter 4, “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*. 
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. The following sections describe basic and common types of cryptography and how they are supported in OpenEdge:

- 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 the “Asymmetric (public) key cryptography” section on page 3–8.

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. Four common algorithms, used by OpenEdge, are:

- **DES** — Data Encryption Standard
- **DES3** — Triple DES
- **AES** — Advanced Encryption Standard
- **RC4** — Also known as ARC4

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.

The RC4 is supplied for situations that call for fast encryption, but not strong encryption. RC4 is ideal for situations that require a minimum of encryption.
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 and RC4)
- 168 (DES3)
- 192 (AES)
- 256 (AES)

The trade-off between the key sizes is that larger keys take longer to encrypt and decrypt.
Symmetric (secret) key cryptography

Combine algorithm, mode, and key size

Symmetric-key algorithms are identified by the three components of their supported variations: \textit{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.

Figure 3–1 shows the relative key strengths on a scale of 1 to 10.

![Figure 3–1: Relative key strength](image)

Figure 3–2 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 3–2: Relative key encryption and decryption speed](image)
OpenEdge usage

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

- To provide Transparent Data Encryption of database blocks. For more information on Transparent Data Encryption, see Part IV, “Transparent Data Encryption.”

- 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 Chapter 5, “Public-Key Infrastructure (PKI)”
  - SSL — See Chapter 5, “Public-Key Infrastructure (PKI)”
  - OpenEdge support for SSL — See Chapter 6, “Secure Sockets Layer (SSL)”

- 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 the “Password-based encryption (PBE)” section on page 3–7)
  - GENERATE-PBE-SALT function (see the “Password-based encryption (PBE)” section on page 3–7)
  - 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 Chapter 5, “Public-Key Infrastructure (PKI)”
- **OpenEdge support for SSL** — See Chapter 6, “Secure Sockets Layer (SSL)”
- **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 Chapter 5, “Public-Key Infrastructure (PKI)”), 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 Chapter 5, “Public-Key Infrastructure (PKI)”
- **SSL** — See Chapter 5, “Public-Key Infrastructure (PKI)”
- **OpenEdge support for SSL** — See Chapter 6, “Secure Sockets Layer (SSL)”
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 sender’s 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.

Common examples of hash algorithms used to generate MAC values include Message Digest 5 (MD5) and Secure Hash Algorithm SHA-1 (SHA-1).

OpenEdge usage

OpenEdge provides message digest features in ABL that you can use to provide your own data integrity tests, including the:

- MD5-DIGEST function
- SHA1-DIGEST function

OpenEdge also allows you to specify the hashing algorithm for PBE (see the “Password-based encryption (PBE)” section on page 3–7) 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 Chapter 5, “Public-Key Infrastructure (PKI).”
- 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 sections on using MD5 message digests in *OpenEdge Deployment: WebClient Applications*.
- Seal audit data from unauthorized alteration. For more information, see Chapter 8, “Auditing in OpenEdge.”
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 Chapter 6, “Secure Sockets Layer (SSL),” in this manual, and the sections on managing OpenEdge key and certificate stores in *OpenEdge Getting Started: Installation and Configuration*. 
A key component of any successful security policy is the management of user identity. The way you approach user identity management determines who can access your application, database, or other resources; what the user can do once access has been given; and even how you will audit the activities that the user performs. In some cases, a user may be an individual; in other cases, a user may be an application or utility of some kind. Either way, controlling unapproved access to data is essential.

OpenEdge provides you with a number of different options regarding how you manage user identities through its authentication, authorization, and user ID assertion services, and how you record user identity for auditing purposes. This chapter provides information about those options and their implementation, as described in the following sections:

- Managing user identity
- Trusted authentication systems and domains
- The client-principal object
- Asserting user IDs individually or in synchronization
- Additional OpenEdge identity management options
Managing user identity

Managing user identity includes, among other things, the processes of authentication and authorization. Authentication is the process of verifying a user’s identity, and authorization is the determination of what data the authenticated user can access and/or which actions and operations the user can perform.

In addition to authentication and authorization of a user, identity management also includes how, where, and when you assert a user ID (in other words, how you use the user ID) once you authenticate it. Table 4–1 lists and describes each user ID type and purpose.

Table 4–1: OpenEdge user identities

<table>
<thead>
<tr>
<th>Identity type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Database connection identity   | A user ID that has been authenticated by the database connection. An OpenEdge RDBMS authorizes all database connections and access to specific database tables and fields using the database connection ID.  
You can set the database connection ID from a user ID that is authenticated using either the OpenEdge internal authentication system (._User table) or an external authentication system. The ABL USERID function returns the current database connection ID for a database connection, regardless of how it is set. |
| OpenEdge session identity      | A user ID that is associated with a given ABL session, independent of any database connections. The OpenEdge session ID can be used to authorize or identify user access to application features in a database-independent fashion. These can be features that are entirely application-defined or that are supported specifically by OpenEdge, such as the auditing identity. You can set the OpenEdge session ID from a user ID that is authenticated using an external authentication system. |
Managing user identity

Authentication in OpenEdge

OpenEdge supports two different methods of user authentication:

- Internal, with the OpenEdge database _User table
- External, with an ABL procedure serving as an authentication system
Using the _User table to establish a database user ID

Authentication to the _User table requires one of the following:

- A user name and password that match those in an existing user account.
- The ABL CONNECT statement.
- The use of the SETUSERID function to authenticate a specified user ID and password to the user IDs and passwords stored in the OpenEdge database. If the user ID is authenticated, this function also asserts the user ID as the current database user ID.

Using an external authentication system

You can set up external authentication systems by setting up your own authentication system implemented through ABL and then configuring OpenEdge to recognize that system. The authentication system implemented by the ABL application can use any source of user accounts, which it can access through an API to an external security system or internally through its own database tables.

The primary components involved in setting up your own ABL-based authentication system are:

- The ABL procedure or procedures that perform the user account validation
- A user login-session object that represents a successfully authenticated user-account and can contain additional account and application data
- A configuration that controls how OpenEdge will validate login-session objects that have been generated by the ABL authentication procedures and set their user IDs as the current OpenEdge session’s default user ID
- A configuration that controls how OpenEdge will validate login-session objects that have been generated by the ABL authentication procedures and set their user IDs as the OpenEdge database’s current user ID

You create and manage user login-session objects through the ABL client-principal object. You pass, or assert, that login-session object to OpenEdge through SECURITY-POLICY methods of ABL functions. OpenEdge AVM sessions and database connections use the configuration information contained in a domain registry to validate the login-session for origin and integrity before using the login-session’s user ID. The domain registry configuration essentially establishes a trust relationship between the ABL procedures that implement an authentication system and OpenEdge, which is responsible for assuring user identity before using it to access application operations and data.

The source of the ABL sessions’s domain registry information can be the ABL application or an OpenEdge database. The source for an OpenEdge database’s domain registry can be its database tables or a copy of the ABL sessions’s domain registry.

Any ABL application can implement, configure, and use any number of user authentication systems and domains, which end users can configure and use at their discretion. For more information, see the “Trusted authentication systems and domains” section on page 4–5 and the “Trusted domain registry” section on page 4–6.
Trusted authentication systems and domains

A trusted authentication system is one that is trusted by a system or application to correctly implement the security processes necessary to ensure that authenticated user accounts are used only by their owners. The trusted authentication system is also expected to assure that it cannot be spoofed into using falsified credentials.

The level of security establishes the amount of trust that may be used to protect application operations and data. Trusted systems have many levels of trust, and an application should use the level of trust appropriate for the value of the application data to the end user. For example, a system based on user names and passwords might be trusted to provide enough security for noncritical information such as warehouse inventory but would not provide enough security to be trusted to protect a company’s financial information.

The OpenEdge database’s sole built-in trusted authentication system is the _User table; once a user is authenticated to the _User table, authorization is administered based on the table and field CAN-* permissions.

**Note:** Table and field CAN-* permissions use a database’s current user ID, which can originate from either the built-in _User table accounts or an external authentication system.

An external trusted authentication system requires you to set up a trust relationship between an ABL application that has its own system user accounts and OpenEdge, so that OpenEdge knows it can trust the user account authentication performed by the ABL application. In OpenEdge, each external authentication system type must be defined before it can be used. That definition involves assigning a common, or logical, name that clearly defines it. For example, LDAP would be a type name for the support an ABL application would provide to use an external LDAP directory service. ABL-Local may be a type name that refers to an ABL application’s procedures that support its own user account tables.

Each instance, or physical implementation, of a trusted authentication system type is known as a trusted authentication domain. After you create an authentication system type, you configure a domain, and provide a name, description, the domain’s access code, and any additional comments you want. The name you provide for the domain is the logical name that will be used by OpenEdge to locate the domain’s configuration at run time to validate the origin and authenticity of the user account that was authenticated by ABL’s authentication system procedures.

When you create a domain, also known as an instance of an authentication system, you define its access code. The access code is stored as an encrypted phrase to prevent its being used to impersonate the legitimate ABL authentication system procedures. The access code functions as a shared secret, much like a password, between ABL procedures that implement the authentication system domain and OpenEdge’s ABL session and/or database connections.

After a successful user authentication, ABL procedures create the login-session token and certify it by performing a seal operation using the domain’s shared secret. Sealing can be likened to the historical practice of sealing an envelope with a wax seal, imprinted with a private stamp, when dispatching important information. If the seal was recognized by the recipient, the recipient knew from whom the envelope and its contents originated. If the seal was unbroken, the recipient also knew that the envelope’s contents had not been tampered with. That same concept exists in today’s security world, and OpenEdge uses it in the trust relationship between ABL-implemented user authentication systems and the OpenEdge run-time system.
**Trusted domain registry**

To configure external authentication systems for run time, you must configure OpenEdge with the authentication system type and domain information it requires to validate the login-session token (client-principal object) generated by the ABL authentication system. This is accomplished by populating an internal storage object referred to as a **trusted domain registry**, or simply a registry. OpenEdge employs multiple registries in its architecture; each registry is capable of being independently configured to protect each ABL session and database connection.

You can combine all the registries together into a single, unified validation source so that no matter where you assert a client-principal object (in a database connection or an ABL session) the validation is performed in the same place. You can also choose to load each registry independently so that a client-principal is validated at each point it is used. The method you choose will reflect the security architecture implemented in your ABL application.

If you want the ABL session to validate client-principal objects at run time to establish a session-wide ID, you need to load the ABL session’s registry through methods provided in the SECURITY-POLICY object. These methods allow the application to load the session’s registry from a trusted OpenEdge database source or manually from any application-supplied source.

Each OpenEdge database also has one trusted **database domain registry**, which is loaded at run time from one of two sources. The registry can be loaded from its database’s tables, which are populated by using either Data Administration or the character Data Dictionary, or it can be loaded from the contents of the ABL session’s registry. The former load method is used when your application wants each OpenEdge database to have a unique configuration of authentication systems and domains for validating client-principal objects. The latter method is used when your application wants to use a single configuration of authentication system and domains that are used by the ABL session and all the OpenEdge database connections.

The interaction and sharing of registry information are controlled through options that are contained in an OpenEdge database and managed using either Data Administration or the character Data Dictionary.

**Populating the registries**

Any one registry can support any number of ABL-supported authentication-type systems. The registry information that is loaded from database tables at run time includes only those authentication domains that are marked as enabled. This provides the flexibility to configure and support multiple types of authentication systems in the ABL application and then simply enable or disable the appropriate one, or ones, at the production site.

Each database has only one registry, but the registry can have multiple domains.

The application registry is populated by:

- Adding external authentication system domains, one at a time, by using the REGISTER-DOMAIN( ) method
- Adding one or more database domain registries by using the LOAD-DOMAINS( ) method
Each registry entry loaded at run time represents a trust relationship, indicating that OpenEdge can trust the authenticated user IDs originating from that ABL authentication system.

Once an application user ID or database connection ID is established by some registry domain entry, it can then be used for various kinds of authorization. The `CAN-*` permissions apply only to database connection IDs; with the `CAN-DO` permissions, it can be any ID that the application wants to use. There is no dependency on having a connection to the `_User` table.

Before the ABL application can assert and validate the user ID, your ABL authentication system procedure must first create the client-principal object. It then inserts the user ID, authentication-domain name, and login-session ID, among other things. When the client-principal object (the user ID) is fully populated, the object may then be sealed to indicate a successful user authentication or failed to indicate a failed user authentication. In both of these cases, the seal and fail methods automatically trigger user login-session auditing if it is enabled.
The client-principal object

Each client-principal object is created and managed by an ABL application and represents a single user's login session, which enables the tracking of a client from logging on to logging out of an ABL session. A client-principal object can be shared for single sign-on purposes, between or among different ABL-type processes. For example, the object can be shared between any of the OpenEdge application servers, or any of the servers' agents. The object can be used to set the current user ID for the ABL application and optionally all of its OpenEdge database connections.

A client-principal object holds the user ID and other relevant user account and application data. It is a transportable user login-session token that the application being accessed can validate, understand, and use in its authorization system to control the ABL application or database connection access to resources. You can use the client-principal object to set the application user ID for the ABL application itself and, globally, the database connection ID from the database connections, as well as individually set a separate database connection ID for each database.

Each client-principal object must be sealed by the ABL user authentication procedure with a domain access code upon login to prove that authentication was successful. The domain access code must be identical to the access code used to register the domain in either the ABL session or OpenEdge database's registry. The client-principal also includes the domain name as a reference to a domain in those registries.

The domain access code is used by OpenEdge to validate the following:

- The client-principal object came from a trusted authentication source.
- The client-principal object has not been tampered with.
- The client-principal object has not expired.
- The client-principal has not been logged out.

If any of these requirements fails, the client-principal object is not honored and is considered unusable; the user ID represented by the client-principal object is not validated or set.

There are two ways for the client-principal object to assert the ID of the user:

- **SECURITY-POLICY: SET-CLIENT( ) method** — Uses the user ID associated with the sealed client-principal to set the default ID for the OpenEdge session and attempts to use the client-principal to set the user ID on all connected OpenEdge databases (that do not already have an ID explicitly set)

- **SET-DB-CLIENT function** — Uses the ID represented by a sealed and validated client-principal object to set a user ID for the specified OpenEdge database
The SET-CLIENT() method and the SET-DB-CLIENT function do not authenticate user IDs; they validate them. When you validate the client-principal object, you are taking an authenticated user ID and making it useful within OpenEdge. If you successfully validate the client-principal object, the user ID is set as an application user ID or a database connection ID. With the SET-DB-CLIENT function, the database connection ID is set; with the SET-CLIENT() method, the application user ID is set. With the SET-CLIENT() method, you can also set the application user ID for a database that does not have a database connection ID explicitly set.

For more information about how to build a client-principal object and assert a user ID, see OpenEdge Development: Programming Interfaces.

**Note:** The client-principal object is not used for data servers.
Asserting user IDs individually or in synchronization

You can decide whether the asserted user ID will be the same for the application and each database connection. There are two basic models that you can use:

- **A single user ID that applies to the entire application and all the database connections.**

  In this case, a single copy of registry information is used by all the domain registries; an ABL database client application uses the same user ID for access control to both the application resources and the database data.

  To do this, the application domain registry must be configured within the ABL session and each database’s registry (or databases’ registries) must be configured to use the ABL session’s domain registry to validate client-principal objects. Within Data Administration, the option must be set for the database to trust the application registry.

  When the database trusts the application domain registry, the database connection trusts and uses that registry for validating application user IDs wanting to access the database. If the option to trust the application domain registry is not set in Data Administration, the registry information is obtained from the database’s internal tables.

  When the database connection attempts to validate a client-principal, whether passed explicitly through the \texttt{SET-DB-CLIENT} function or implicitly through the \texttt{SET-CLIENT} method, it will use the contents of the application’s registry to perform the validation on the client-principal.

  The \texttt{SET-DB-CLIENT} function and the \texttt{SETUSERID} function always override any database connection ID set through the \texttt{SET-CLIENT} method.

- **Multiple user authentication where the user identity for the application and each database connection can be set individually.**

  To do this, each database registry must be configured (within Data Administration) with the domains that can be trusted. Optionally, the ABL session’s domain registry may be configured for use in independently validating user IDs at the ABL application level.

  The configuration process is more complex with this option, in which applications use more than one user ID; the user ID used to access the database is not the same one used to control access from within the application. You can configure the application domain registry and each database connection to use its own registry.
Additional OpenEdge identity management options

In addition to deciding how to manage user identity and authentication, you can also choose to do the following:

- Record authenticated client sessions
- Disallow database connections with a blank user ID

You make each of these selections from Data Administration. For additional details, see the Data Administration Help.

Recording authenticated client sessions

You can optionally record to the database the detailed login information that fully describes where and how a user logs in (using a client-principal object). If you do not select this option, you get no client-session context.

**Note:** Login, failure, and logout are automatically audited if you have auditing turned on.

For details about selecting this option, see the Data Administration Help.

Disallowing the blank user ID

You can prevent the blank or default user ID from connecting to an OpenEdge database when connecting with a Release 10.1 or later client.

You select the option to prevent blank user IDs from accessing the database by choosing **Admin → Database Options → Disallow BlankUserId** in Data Administration. For details, see the Data Administration Help.

**Note:** Choosing the **Disallow Blank UserId** option from the **Admin → Database Options** menu is different from choosing **Disallow Blank Userid Access** from the **Admin → Security** menu. In the latter case, you are allowing the user to access fields and tables in the database as long as the field- and table-level permissions allow it and the procedures being run are precompiled.
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.

This chapter summarizes these basic PKI concepts in the following sections:

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

**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.
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
- Confidentiality
- Integrity

For basic information on these security services, see the sections on security in Chapter 1, “OpenEdge Core Business Services.” A PKI supports all of these services using cryptographic mechanisms that serve as guarantors of the assurances provided by these services (see the “Cryptography in a PKI” section on page 5–4).

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 the “Cryptography in a PKI” section on page 5–4.

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 the “Cryptography in a PKI” section on page 5–4.
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 the “Confidentiality” section on page 5–2), 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 Chapter 3, “Cryptography.”

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
- Message digests and MACs
- Public-key cryptography
- Password-based encryption (PBE)

Symmetric-key cryptography

The foundation of all PKI is public-key (also known as asymmetric-key) cryptography (see the “Public-key cryptography” section on page 5–5), 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 the “Password-based encryption (PBE)” section on page 5–5). 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 the “Trust relationships and supporting mechanisms” section on page 5–6.
Cryptography in a PKI

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 the “Trust relationships and supporting mechanisms” section on page 5–6.

- **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” section on page 5–6.

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 owner’s 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 Chapter 1, “OpenEdge Core Business Services.”

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 CA’s 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

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 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.

**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 certificate’s lifetime has expired, it can no longer be used to assert or authenticate a server’s identity.

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 owner’s identity, must be submitted to a CA that validates the owner’s identity and, if valid, issues a digital certificate that contains the owner’s public key. The location for storing the server’s private key is commonly known as a **key store**.

A key store must allow the owner to manage the server’s 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 identity 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 client’s 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 CA’s 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.
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.

The topics described in this chapter include:

- 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:

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

For an overview of Public-Key Infrastructure, see Chapter 5, “Public-Key Infrastructure (PKI).”
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, see Chapter 7, “SSL in OpenEdge.”

- **PKI support** — Provides the ability for an SSL client to validate an SSL server’s 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 server’s 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 CA’s 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 the “Session caching” section on page 6–5). 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**
- **Keys and certificates**
- **Algorithms (ciphers)**
- **Session caching**

**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** — Electronic documents used to store and access public keys and information for SSL client and server authentication.
- **“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.
SSL session components

- **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

**Notes:** 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 the “Keys and certificates” section on page 6–4). 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 the “SSL standards support in OpenEdge” section on page 6–2), 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 Chapter 7, “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, as outlined in the following sections:

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

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.
OpenEdge SSL architecture

Figure 7–1 shows the general support for SSL in OpenEdge.

Figure 7–1: SSL architecture in OpenEdge (See Figure 7–2 for key to diagram.)
Figure 7–2 provides the key to the architectural diagram shown in Figure 7–1.

<table>
<thead>
<tr>
<th>Key to OpenEdge SSL architecture diagrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄 OpenEdge -managed SSL client connection points</td>
</tr>
<tr>
<td>🔄 OpenEdge -managed SSL server connection points</td>
</tr>
<tr>
<td>🔄 Platform-managed SSL client connection points on the Internet (HTTPS)</td>
</tr>
<tr>
<td>🔄 Platform-managed SSL server connection points on the Internet (HTTPS)</td>
</tr>
<tr>
<td>🔄 Communications to or from the Internet (HTTPS)</td>
</tr>
</tbody>
</table>

**Figure 7–2:  Key to OpenEdge SSL architecture in Figure 7–1**

In Figure 7–1, 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.

**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 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 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 the “Managing SSL server identity” section on page 7–7.

**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 7–1 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 Chapter 5, “Public-Key Infrastructure (PKI).”
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 server’s key store. Each SSL client must successfully validate the server’s identity using a corresponding root public-key certificate (root certificate store entry) accessed from the client’s 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 Microsoft’s 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 client’s 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 client’s 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
- Configuring SSL client sessions
- Accessing an SSL server’s X.500 Subject Name

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
- Progress Explorer 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

Table 7–1 describes the SSL session properties that you can set for an OpenEdge SSL server.

Table 7–1:  OpenEdge server SSL session properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Default setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL connections enabled</td>
<td>No</td>
<td>Specifies if all connections to this server must use SSL.</td>
</tr>
<tr>
<td>Key alias name</td>
<td>“default_server”</td>
<td>Sets the alias name of the key store entry used by the SSL server to assert its identity to the SSL client attempting to connect to that server.</td>
</tr>
<tr>
<td>Key alias password</td>
<td>20333c34252a2137 (which is “password”, encrypted)</td>
<td>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 the Progress 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 the “Managing your own SSL server identity” section on page 7–8).</td>
</tr>
<tr>
<td>Use a session cache</td>
<td>Yes</td>
<td>Specifies if the server provides a cache for SSL client sessions that might be resumed after the client disconnects from the SSL server.</td>
</tr>
<tr>
<td>Session time-out</td>
<td>180 seconds</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 7–2 describes the mechanisms for setting SSL session properties for OpenEdge SSL servers and indicates where you can find more information about them.

Table 7–2: Mechanisms for setting OpenEdge SSL server session properties

<table>
<thead>
<tr>
<th>This SSL server component . . .</th>
<th>Relies on this mechanism . . .</th>
<th>To set its SSL properties as described in . . .</th>
</tr>
</thead>
</table>
| OpenEdge RDBMS                  | Startup parameters              | • OpenEdge Data Management: Database Administration  
                                       • OpenEdge Deployment: Startup Command and Parameter Reference |
| AppServer                       | The Progress Explorer framework | • OpenEdge Application Server: Administration  
                                       • Progress Explorer online help |
| SonicMQ BrokerConnect           | The Progress Explorer framework | • OpenEdge Application Server: Administration  
                                       • Progress Explorer online help |
| WebSpeed Transaction Server     | The Progress Explorer framework | • OpenEdge Application Server: Administration  
                                       • Progress 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

Table 7–3 describes the SSL session properties that you can set for an OpenEdge SSL client.

Table 7–3: OpenEdge client SSL session properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Default setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use SSL to connect</td>
<td>No</td>
<td>Specifies if the client connection uses SSL tunneling.</td>
</tr>
<tr>
<td>Reuse a disconnected SSL session</td>
<td>Yes</td>
<td>Specifies if the connection reuses a previously established and disconnected SSL session when the client connects to an SSL server.</td>
</tr>
<tr>
<td>Verify the SSL server host name</td>
<td>Yes</td>
<td>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.</td>
</tr>
<tr>
<td>Certificate store location (Java Open Client only)(^1)</td>
<td>None</td>
<td>Specifies the location of the root certificate store used by the client for SSL connections.</td>
</tr>
</tbody>
</table>

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.
Table 7–4 describes the mechanisms for setting SSL session properties for OpenEdge SSL clients and indicates where you can find more information about them.

<table>
<thead>
<tr>
<th>This SSL client component . . .</th>
<th>Relies on this mechanism . . .</th>
<th>To set its SSL properties as described in . . .</th>
</tr>
</thead>
</table>
| ABL database client             | Startup and connection parameters | • OpenEdge Deployment: Startup Command and Parameter Reference  
• OpenEdge Development: Programming Interfaces  
• OpenEdge Development: ABL Reference |
| ABL AppServer client¹           | 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 services¹ | Connection parameters | • OpenEdge Development: Web Services  
• OpenEdge Development: ABL Reference |
| ABL client of the SonicMQ BrokerConnect¹ | 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 client¹              | Connection parameters | • OpenEdge Development: Programming Interfaces  
• OpenEdge Development: ABL Reference |
| .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) |
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 server’s 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 7–3), 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.

Table 7–5 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 7–5: Mechanisms for accessing the SSL server X.500 Subject name

<table>
<thead>
<tr>
<th>This SSL client component . . .</th>
<th>Uses this mechanism . . .</th>
<th>To access the SSL server name as described in . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABL database client</td>
<td>SSL-SERVER-NAME function</td>
<td>• OpenEdge Development: ABL Reference</td>
</tr>
</tbody>
</table>
| 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 |
Part III

Auditing

Chapter 8, Auditing in OpenEdge
Chapter 9, Audit Security
Chapter 10, Configuring OpenEdge Auditing
Chapter 11, Developing an Audit-enabled OpenEdge Application
Chapter 12, Deploying an Audit-enabled OpenEdge Application
Chapter 13, Maintaining Audit Data
Chapter 14, Querying and Reporting on Audit Data
This chapter introduces the OpenEdge auditing core service. Specifically, the chapter provides details about the following topics:

- OpenEdge auditing
- Evaluating your auditing requirements
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 database’s 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 the “Setting up OpenEdge auditing context” section on page 10–5.

**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 Appendix B, “Preconfigured Audit Policies,” and the Audit Policy Maintenance Help.

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 application’s database events are recorded or in a database used specifically for application events; or you can record the events in any combination of an application’s 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 Chapter 10, “Configuring OpenEdge Auditing,” Chapter 14, “Querying and Reporting on Audit Data,” 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 application’s 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 application’s 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 application’s audit events with those recorded by the OpenEdge database. Direct the application’s 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.*
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.

This chapter describes the following OpenEdge auditing security features:

- 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
- 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.

Instead of relying solely on the OpenEdge _User table for authentication, you can now establish an ABL application as a trusted source of user authentication. The ABL application will then be able to use its own authentication system, and OpenEdge will accept as authentic all user IDs from that source.

For more information, see Chapter 4, “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 database administrator

When you initially install and start OpenEdge, it is the database 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 database administrator can perform all audit policy functions.

Once the database administrator grants another user audit administrator privileges, however, the database administrator is no longer able to maintain the audit and policy data. The database administrator still retains control over the database’s physical structure. For example, the database 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 database administrator is responsible for access control security for the general application data and has the ability to control user access and in most cases the user accounts themselves.

In some cases, it might be that the audit administrator and the database administrator are the same person; however, in a case in which there is one audit administrator and one or more database 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 Table 9–1, 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 9–1: Granting audit privileges to other users**

<table>
<thead>
<tr>
<th>A user with this audit privilege . . .</th>
<th>Can grant this privilege to other users . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audit administrator</td>
<td>Audit administrator</td>
</tr>
<tr>
<td>Application audit event inserter</td>
<td>Application audit event inserter</td>
</tr>
<tr>
<td>Audit data reporter</td>
<td>Audit data reporter</td>
</tr>
<tr>
<td>Audit data archiver</td>
<td>Audit data archiver</td>
</tr>
<tr>
<td>Application audit event inserter</td>
<td>Audit data archiver</td>
</tr>
<tr>
<td>Audit data reporter</td>
<td>Audit data archiver</td>
</tr>
<tr>
<td>Audit data archiver</td>
<td>Audit data archiver</td>
</tr>
</tbody>
</table>

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 database administrator or ABL administrator automatically inherits the audit administrator privilege. If no specific database administrator or ABL administrator is defined, all users are, effectively, database 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 database 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 administrator’s unforeseen unavailability or absence.

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

A database administrator grants the initial audit privileges; the administrator inherits this responsibility simply by being a database 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 database administrator or a SQL administrator.

At the time the first audit administrator is created, any audit privileges granted to user accounts by the database 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 a SQL administrator, 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 database administrator grants audit administrator privileges to a user, the database administrator no longer has audit administrator privileges. The exception to this is when there is only one audit administrator, in which case the database administrator can revoke that user’s audit administrator privileges (in the case of an emergency). This allows the database 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 database administrator can revoke the user’s audit administrator privileges, the database administrator cannot revoke any of that user’s other permissions.
Granting audit privileges in SQL

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

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

- The SQL product’s database administrator will retain the ability to SELECT (read) on all audit policy and databases.
- The SQL database administrator 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 user’s 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 A’s audit data reporter privilege is later revoked, the same will be true for users B, C, and D. However, if user A’s 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 B’s audit data reporter privilege, for example, but allow user C to retain that same privilege.

**Note:** Changes to a user account’s 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 database’s 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.

- 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 database 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 instance’s 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 database 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 table’s 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 connection’s 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 user’s 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.

Previously, the only trusted user ID source for OpenEdge was the _User table. However, reliance on the _User table meant that it was not possible for OpenEdge to validate that a user ID from an ABL application was coming from a trusted ABL application 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 allow an ABL application to become a trusted source of user authentication. With the availability of the ABL client-principal object and Trusted Application Domain Registry, OpenEdge can establish a trust relationship with an ABL application; the ABL application will then be able to use its own authentication system, and OpenEdge will be able to accept as authentic all user IDs from that source.

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, the _User table is still a requirement for authentication in SQL.

If you are connected as a database administrator, you can now do the following:

- Define authentication type or types implemented within your ABL applications
- Define authentication system domains that execute a particular authentication system at run time with your ABL applications
For the ABL application to use its own authentication system (such as its own version of a _User table, perhaps), you must also make sure that:

- The domain that originates a client-principal object is included in the trusted authentication domain registry
- The client-principal object is sealed, denoting a successful user authentication

For more information, see Chapter 4, “Identity Management,” and *OpenEdge Development: Programming Interfaces.*

**Notes:** If you continue to use the OpenEdge _User table for authentication, you need not do anything further to set up authentication. Connecting to an OpenEdge database by using either no ID or the user parameter (-U) and the password parameter (-P) is audited automatically when you have the audit service enabled.
Controlling the user ID and login sessions in audit data records

As described in the “Configuring additional user authentication systems and domains” section on page 9–16 and in Chapter 4, “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 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 existing model, your application is required to connect to an OpenEdge database using either the -U and -P options or a blank user ID connection, 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 user authentication-system domain is always OpenEdge.

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

1. Coding the ABL user authentication system that will generate the client-principal objects into the application.
2. Configuring the Trusted Domain Registry records in the OpenEdge database or databases.
3. Configuring the user authentication database options to set the authentication trust relationships. For example:
   - The database connection requires that a user ID (client-principal) be validated against the database's Trusted Domain Registry
   - The database connection trusts a user ID (client-principal) that has been validated by the application’s Trusted Domain Registry
4. Coding the ABL application's initialization and use of an application-level Trusted Domain Registry. For example:
   - Loading the Trusted Domain Registry from a specific OpenEdge database's tables
   - Loading the Trusted Domain Registry from an external, ABL-brokered source

5. Coding the ABL application points where a user ID must be set. For example:
   - Using the `SECURITY-POLICY:SET-CLIENT( )` method to validate and set an application-wide user ID, including all database connections
   - Using the `SET-DB-CLIENT` function to explicitly validate and set a user ID for a specific OpenEdge database connection (each OpenEdge database connection must be set individually)

Whether the application uses a single user ID (client-principal) for the entire application or multiple user IDs is totally at the discretion of the application designer. The one rule is that the `SET-DB-CLIENT( )` function supersedes any user ID 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 user authentication system that was used.

The recorded user ID in the auditing records can come from either the OpenEdge session for those applications that provide their own user authentication and authorization or the OpenEdge database connection for those applications that use OpenEdge user authentication and authorization. You control where the user ID comes from through configuration options contained in the OpenEdge database and which 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 is set through one of the following three methods:

- The OpenEdge database connection `-u` option
- 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 OpenEdge session, which is controlled through the `SECURITY-POLICY` object. On that object is a method `SET-CLIENT( )`, which allows setting the session-wide user ID by using a client-principal object.

For more information about:

- Managing user IDs, see Chapter 4, “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 Security

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 database’s 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 system’s 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 file’s contents can be validated before audit data is physically loaded into an OpenEdge database’s 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 database’s _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 administrator’s 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.
Configuring OpenEdge Auditing

This chapter is intended to help you and your customer production sites get up and running with auditing, as described in the following sections:

- 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):

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

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 OpenEdge Release 10.1.

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 Appendix B, “Preconfigured Audit Policies.”

   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. Figure 10–1 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 10–1: Auditing context levels**

**Database transaction context**

As shown in Figure 10–1, at the lowest level exists the database transaction context. The database transaction context identifies all audit events that are related to a single client’s database transaction. The database transaction context’s scope is indirectly controlled by the client application’s 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 client’s 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 user’s 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 database’s _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 object’s 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 event’s configuration in the database’s audit policy. During this process, the primary index ID remains the same regardless of which database the event is recorded into. Even though each database’s 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 client’s database transaction ID.
Figure 10–2 shows the relationship among the various auditing records.

Each client’s 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 context’s 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 connection’s 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 context’s 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 application’s 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 application’s 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 application’s 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 database’s 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 application’s 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 application’s 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.
Developing an Audit-enabled OpenEdge Application

In the following sections, this chapter describes how to develop an audit-enabled OpenEdge application:

• 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 PROUTIL tool 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 database’s structure file), see the “Getting started” section on page 10–2.

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 the “Setting up OpenEdge auditing context” section on page 10–5.
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 application’s 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 database’s 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 database’s 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 database’s _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 application’s 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 application’s user accounts and authenticating the application as one of those users, you must perform the following changes:

- 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 Chapter 4, “Identity Management,” for information about establishing an OpenEdge application’s 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 record’s 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 application’s 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 Chapter 9, “Audit Security.”
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 schema’s 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 developer’s 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 application’s 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 application’s 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.
Deploying an Audit-enabled OpenEdge Application

This chapter describes how to deploy an audit-enabled application, as detailed in the following sections:

- 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 site’s 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 application’s 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 the “Audit-enabling your OpenEdge application” section on page 11–3 for 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 application’s 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 the “Loading audit event definitions” section on page 12–3 for audit event definitions. After the policy templates are loaded, you can use Audit Policy Maintenance or the vendor’s 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. Go to Step 2.
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 Appendix A, “Audit Data Tables.”

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 application’s 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-customer’s 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 database’s 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-customer's 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 database's 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.
Maintaining Audit Data

An important part of developing and implementing a successful auditing solution is maintaining the generated audit data. This chapter describes audit data maintenance in 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 database’s 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 Chapter 14, “Querying and Reporting on Audit Data.”
Querying and Reporting on Audit Data

This chapter describes some of the intricacies of the auditing data and factors to be aware of as you query the data, as detailed in the following sections:

- 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 the “Reporting on audit data with prefiltered and custom reports” section on page 14–17.

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

Figure 14–1 shows an entity relationship diagram for the audit data and supporting tables and provides a brief description of each from a reporting perspective.

Figure 14–1: Audit tables schema
Note the following points regarding Figure 14–1:

- The primary table is _aud-audit-data, which holds 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 stream’s 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`) 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 as 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.

The following example is an event context format for internal database PROUTIL utility events:

```
<db-name>.<owner>.<table>
```

All database event record operations use the following format:

```
<owner>.<table>chr(6)<id-fld-val>[chr(7)<id-fld-val>chr(6)<id-fld-val>…]
```

The `id-fld-val` refers to the identifying field values as defined in the table policy records, defaulting to a table’s 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 client’s 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 the “Reporting modified field old/new values” section on page 14–10.

Reporting by audit date and time

Note that the date and time recorded in the audit records is the database’s date and time rather than the client’s 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 Table 14–1.

Table 14–1: Sample data type values

<table>
<thead>
<tr>
<th>Data type</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Character</td>
</tr>
<tr>
<td>2</td>
<td>Date</td>
</tr>
<tr>
<td>3</td>
<td>Logical</td>
</tr>
<tr>
<td>4</td>
<td>Integer</td>
</tr>
<tr>
<td>5</td>
<td>Decimal</td>
</tr>
<tr>
<td>7</td>
<td>Recid</td>
</tr>
<tr>
<td>8</td>
<td>Raw</td>
</tr>
<tr>
<td>11</td>
<td>Memptr</td>
</tr>
<tr>
<td>13</td>
<td>Rowid</td>
</tr>
<tr>
<td>18</td>
<td>BLOB</td>
</tr>
<tr>
<td>19</td>
<td>CLOB</td>
</tr>
<tr>
<td>34</td>
<td>Datetime</td>
</tr>
<tr>
<td>39</td>
<td>Longchar</td>
</tr>
<tr>
<td>40</td>
<td>Datetime-TZ</td>
</tr>
</tbody>
</table>

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:

\[
E[8]:279.67\text{chr}(8)E[11]:0
\]

\(E[n]\) means the \(n\)th element in the array; the array index starts from 1.

When recording modified field values in the parent record in 
\texttt{_aud-audit-data._event-detail}, the structure of the data is as follows:

\[
\text{fieldname + chr(6) + data type code + chr(6) + old value + chr(6) + new value + chr(7)}
\]

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 \texttt{_event-detail} field is and whether to expect any child records in the \texttt{_aud-audit-data-value} table.

One way is to check the event definition and rely on the event name ending in \texttt{.create} or \texttt{.delete} or \texttt{.update}. All the internal policies conform to this. This can be checked in code as follows:

```
DEFINE VARIABLE tDbEvent 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 tDbEvent = 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-trg, _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 see Appendix B, “Preconfigured Audit Policies.”

**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 Table 14–2 (assuming indexes are enabled).

**Table 14–2: _aud-audit-data main table indexed fields**

<table>
<thead>
<tr>
<th>Index name</th>
<th>Flags</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Data-guid</td>
<td>primary</td>
<td>_Audit-data-guid</td>
<td>Primary unique index. Joins the child table _Aud-daudit-data-value that holds the modified field’s old and new values. This index is always active.</td>
</tr>
<tr>
<td>_Connection-id</td>
<td>_</td>
<td>_Database-connection-id</td>
<td>Queries audit data by database connection, and further within a database connection by client session.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Client-session-uuid</td>
<td></td>
</tr>
<tr>
<td>_Userid</td>
<td>_</td>
<td>_User-id</td>
<td>Queries audit data by the real application user who created the audit data.</td>
</tr>
<tr>
<td>_Event-group</td>
<td>_</td>
<td>_Audit-event-group</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Db-guid</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Transaction-id</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Transaction-sequence</td>
<td></td>
</tr>
<tr>
<td>_EventId</td>
<td>_</td>
<td>_Event-id</td>
<td>Queries audit data by audit event ID; event IDs between the range 5000 to 5099 represent schema change events.</td>
</tr>
<tr>
<td>_Audit-time</td>
<td>_</td>
<td>_Audit-date-time</td>
<td>Queries audit data by date and time it was created (plus time zone). This index is always active.</td>
</tr>
</tbody>
</table>
The indexed fields on the _aud-audit-data-value child table are listed in Table 14–3.

**Table 14–3: _aud-audit-data value child table indexed fields**

<table>
<thead>
<tr>
<th>Index name</th>
<th>Flags</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Continuation-seq</td>
<td>Pu</td>
<td>_Audit-data-guid</td>
<td>The primary unique index for the table; used when joining from the parent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Field-name</td>
<td>_Aud-audit-data table to determine the modified fields for a particular audit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Continuation-sequence</td>
<td>data entry.</td>
</tr>
<tr>
<td>_Field-name</td>
<td>–</td>
<td>_Field-name</td>
<td>Facilitates querying audit data for changes to a specific field.</td>
</tr>
</tbody>
</table>

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 primary key. 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-table fields 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** — 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 in Table 14–4.

- **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 report’s page length is fixed or continuous.

Figure 14–2 shows a report in detail orientation.

![Figure 14–2: Detail orientation report output](image)
Figure 14–3 shows a report in summary orientation.

![Figure 14–3: Summary orientation report output](image)

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.

Table 14–4 lists each of the prefiltered reports and describes the information the report contains.

<table>
<thead>
<tr>
<th>Report name</th>
<th>Description</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Track Audit Policy Changes</td>
<td>Reports on audit policy changes</td>
<td>• Audit Policy Maintenance Events: record created, updated, or deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Audit Policy Commit Events: audit policy run-time update/commit changes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Audit Policy Dump or Load Events: dump and load of audit policies</td>
</tr>
</tbody>
</table>
### Prefiltered audit data reports

<table>
<thead>
<tr>
<th>Report name</th>
<th>Description</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Track Database Schema Changes</td>
<td>Reports on all possible schema modification actions</td>
<td>• Table created, updated, or deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Table trigger created, updated, or deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Table field created, updated, or deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Table field trigger created, updated, or deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Table index created, updated, or deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Table index field created, updated, or deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Sequence created, updated, or deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Database property created or updated</td>
</tr>
<tr>
<td>Track Audit Data Administration (Dump/Load)</td>
<td>Reports on all archiving and loading of audit data</td>
<td>Audit archive or audit data load</td>
</tr>
<tr>
<td>Track Application Data Administration (Dump/Load)</td>
<td>Reports on archiving and loading of audit data</td>
<td>Database text archive or load</td>
</tr>
<tr>
<td>Track User Account Changes</td>
<td>Reports on user account activity</td>
<td>User account created, updated, or deleted</td>
</tr>
<tr>
<td>Track Security Permission Changes</td>
<td>Reports on privilege definition and privilege assignment activity</td>
<td>• Privilege definition created, updated, or deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Privilege assignment created, updated, or deleted</td>
</tr>
<tr>
<td>Track SQL Permissions Changes</td>
<td>Reports on all possible SQL privilege modifications</td>
<td>• SQL database administrator created, updated, or deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SQL table privilege created, updated, or deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SQL column privilege created, updated, or deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SQL sequence privilege created, updated, or deleted</td>
</tr>
</tbody>
</table>
### Table 14–4: Prefiltered audit data reports

<table>
<thead>
<tr>
<th>Report name</th>
<th>Description</th>
<th>Events</th>
</tr>
</thead>
</table>
| 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.

Figure 14–4 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.

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 Figure 14–5.

![Custom audit data filter report detail](image)

**Figure 14–5**: Custom audit data filter report detail

You can review the details about each record and sort through the records in the order you want.
Part IV

Transparent Data Encryption

Chapter 15, Transparent Data Encryption in OpenEdge RDBMS
Chapter 16, OpenEdge Key Store
Chapter 17, Configuring Transparent Data Encryption policies
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. Transparent Database Encryption concepts are discussed in the following sections:

- 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 database’s 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 Chapter 16, “OpenEdge Key Store” 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 database’s 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 Chapter 17, “Configuring Transparent Data Encryption policies.”

Transparent data encryption depends on secure storage of encryption keys, and user access controls to your encryption keys to ensure that your data’s 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 Chapter 16, “OpenEdge Key Store.”
Data security

Data security as pertains to Transparent Data Encryption applies in many areas, as discussed in the following sections:

- Data Storage
- Encrypted Data Configuration
- Encryption Key

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 database’s 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 database’s 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 runtime 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 Chapter 16, “OpenEdge Key Store.”
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 Chapter 3, "Cryptography." For a list of the object ciphers supported for Transparent Data Encryption, see the Table 17–1.

- **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 the Chapter 16, “OpenEdge Key Store” 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 Chapter 17, “Configuring Transparent Data Encryption policies.”

**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

**Notes:** 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 table’s 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 database’s 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
- Executing the ENABLEENCRYPTION command

Notes: 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:

**Syntax**

```
  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 Chapter 3, “Cryptography.” Table 15–1 lists the supported database master key ciphers.

<table>
<thead>
<tr>
<th>ID</th>
<th>Cipher</th>
<th>Mode</th>
<th>Size</th>
<th>Key type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AES</td>
<td>CBC</td>
<td>128</td>
<td>binary</td>
</tr>
<tr>
<td>2</td>
<td>AES</td>
<td>CBC</td>
<td>192</td>
<td>binary</td>
</tr>
<tr>
<td>3</td>
<td>AES</td>
<td>CBC</td>
<td>256</td>
<td>binary</td>
</tr>
<tr>
<td>4</td>
<td>DES</td>
<td>CBC</td>
<td>56</td>
<td>binary</td>
</tr>
<tr>
<td>5</td>
<td>DES3</td>
<td>CBC</td>
<td>168</td>
<td>binary</td>
</tr>
<tr>
<td>6</td>
<td>DES</td>
<td>CBC</td>
<td>56</td>
<td>PBE</td>
</tr>
<tr>
<td>7</td>
<td>RC4</td>
<td>ECB</td>
<td>128</td>
<td>binary</td>
</tr>
</tbody>
</table>
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 Table 15–2.

Table 15–2: Passphrase constraints

<table>
<thead>
<tr>
<th>Rule</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum number of characters</td>
<td>8</td>
</tr>
<tr>
<td>Maximum number of characters</td>
<td>2048</td>
</tr>
<tr>
<td>Minimum number of numeric characters</td>
<td>1</td>
</tr>
<tr>
<td>Minimum number of alpha characters</td>
<td>2</td>
</tr>
<tr>
<td>Minimum number of punctuation characters</td>
<td>1</td>
</tr>
<tr>
<td>Character set</td>
<td>[a-zA-Z0-9]!@#$%^&amp;*()_+-{}[]\</td>
</tr>
<tr>
<td>First character</td>
<td>(see Character set)</td>
</tr>
<tr>
<td>Mixed case alpha required</td>
<td>True</td>
</tr>
<tr>
<td>Case sensitive</td>
<td>True</td>
</tr>
</tbody>
</table>

Once you have enabled your database for Transparent Data Encryption, you must define encryption policies before any data is encrypted. See Chapter 17, “Configuring Transparent Data Encryption policies” for details.
The OpenEdge key store provides secure storage of encryption keys in support of Transparent Data Encryption. The OpenEdge key store and general key store concepts are discussed in the following sections:

- 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 key’s value
- The user access controls that limit the encryption key’s 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 store’s 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 the “Enabling Encryption” section on page 15–11. 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 passphrase 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:

<table>
<thead>
<tr>
<th>size</th>
<th>area name</th>
<th>path name</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Primary Recovery Area</td>
<td>C:\OE102BWrk\dest_db.b1 00:00:00</td>
</tr>
<tr>
<td>16</td>
<td>Schema Area</td>
<td>C:\OE102BWrk\dest_db.d1 00:00:00</td>
</tr>
<tr>
<td>16</td>
<td>Info Area</td>
<td>C:\OE102BWrk\dest_db_7.d1 00:00:00</td>
</tr>
<tr>
<td>16</td>
<td>Customer/Order Area</td>
<td>C:\OE102BWrk\dest_db_8.d1 00:00:00</td>
</tr>
<tr>
<td>16</td>
<td>Primary Index Area</td>
<td>C:\OE102BWrk\dest_db_9.d1 00:00:00</td>
</tr>
<tr>
<td>64</td>
<td>Customer Index Area</td>
<td>C:\OE102BWrk\dest_db_10.d1 00:00:01</td>
</tr>
<tr>
<td>64</td>
<td>Order Index Area</td>
<td>C:\OE102BWrk\dest_db_11.d1 00:00:00</td>
</tr>
<tr>
<td>256</td>
<td>Encryption Policy Area</td>
<td>C:\OE102BWrk\dest_db_12.d1 00:00:00</td>
</tr>
<tr>
<td>64</td>
<td>Encryption Policy Area</td>
<td>C:\OE102BWrk\dest_db_12.d2 00:00:00</td>
</tr>
<tr>
<td>16</td>
<td>Audit Area</td>
<td>C:\OE102BWrk\dest_db_20.d1 00:00:00</td>
</tr>
<tr>
<td>16</td>
<td>Audit Index Area</td>
<td>C:\OE102BWrk\dest_db_22.d1 00:00:00</td>
</tr>
</tbody>
</table>

Copying src_db to dest_db... (6715)

Start writing data blocks. (6718)
01:15:53 10 Percent complete.
01:15:53 20 Percent complete.
01:15:53 30 Percent complete.
01:15:53 40 Percent complete.
01:15:53 50 Percent complete.
01:15:53 60 Percent complete.
01:15:53 70 Percent complete.
01:15:53 80 Percent complete.
01:15:53 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 successful 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>
```
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. Transparent Data Encryption policies are discussed in the following sections:

- 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 the “Encrypting your existing data” section on page 17–6 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 the “Creating encryption policies with PROUTIL EPOLICY” section on page 17–2.
- To create a policy with Data Admin, see the “Creating encryption policies with Data Admin” section on page 17–3.
- To create a policy with OpenEdge SQL, see the “OpenEdge SQL support for Transparent Data Encryption” section on page 17–13.

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:

Syntax

```
proutil db-name -C epolicy manage object-type encrypt object-name
```

OpenEdge supports the object ciphers described in the Table 17–1.

Table 17–1: Object ciphers

<table>
<thead>
<tr>
<th>ID</th>
<th>Cipher</th>
<th>Mode</th>
<th>Size</th>
<th>Key type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NULL</td>
<td>NULL</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>AES</td>
<td>CBC</td>
<td>128</td>
<td>binary</td>
</tr>
<tr>
<td>2</td>
<td>AES</td>
<td>CBC</td>
<td>192</td>
<td>binary</td>
</tr>
<tr>
<td>3</td>
<td>AES</td>
<td>CBC</td>
<td>256</td>
<td>binary</td>
</tr>
<tr>
<td>4</td>
<td>DES</td>
<td>CBC</td>
<td>56</td>
<td>binary</td>
</tr>
<tr>
<td>5</td>
<td>DES3</td>
<td>CBC</td>
<td>168</td>
<td>binary</td>
</tr>
<tr>
<td>7</td>
<td>RC4</td>
<td>ECB</td>
<td>128</td>
<td>binary</td>
</tr>
</tbody>
</table>

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.
Creating encryption policies

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 Data Admin

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 the “Creating encryption policies with PROUTIL EPOLICY” section on page 17–2 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 Data Administration, choose Admin → Security → Encryption Policies → Edit Encryption Policy. The Object Selector dialog appears:

   ![Object Selector dialog](image)

   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:

![Edit Encryption Policy dialog]

Note: Passphrase and Verify Passphrase are always disabled.

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:

![Copy Encryption Policy Setting To dialog](image)

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:

![Review changes](image)

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:

![Commit confirmation](image)

---

**Note:** Editing encryption policies is fully supported in the character Data Dictionary.
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.
All new data is encrypted

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. Figure 17–1 depicts the periodic updating of encryption policies in the life cycle of encrypted data.

Figure 17–1: 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 the “Rekey encryption policies with Data Admin” section on page 17–10.
- To rekey a policy from the command line, see the “Rekey an encryption policy with PROUTIL EPOLICY” section on page 17–11.
- To rekey a policy with OpenEdge SQL, see the “OpenEdge SQL support for Transparent Data Encryption” section on page 17–13.

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 the “Viewing encryption policy history in Data Admin” section on page 17–12 for instructions.
Rekey encryption policies with Data Admin

Generate new encryption keys for encryption policies in your encryption-enabled database with the Generate Encryption Keys dialog. 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 with Data Admin.

The following steps present a guide for generating new encryption keys.

To generate a new encryption key for an existing encryption policy:

1. From Data Administration, choose Admin→Security→Encryption Policies→Generate Encryption Keys. The Object Selector dialog appears:

   ![Object Selector](image)

   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:

![Generate Encryption Keys dialog](image)

**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.

![Commit dialog](image)

**Note:** Generating new encryption keys is fully supported in the character Data Dictionary.

**Rekey 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.
**Change the cipher of and 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 Data Admin**

You can view the history of an encryption policy in Data Administration. Use the following steps to view encryption policy history for encrypted objects in an encryption enabled database.

To view encryption policy history:


![Encryption Policy History dialog](image)

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:

Syntax

```
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 ]
;
```

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:

Syntax

```plaintext
CREATE TABLE
/
[ 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 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:

Syntax

```plaintext
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:

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 DECRIPT
    |    | 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 DECRIPT
    |    | 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:

**Syntax**

```sql
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
- Object’s 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:

**Syntax**

```
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 DISABLEEENCYPTION at anytime during the execution of the command. Cancelling DISABLEEENCYPTION does not affect the integrity and usability of the database, however cancelling does not revert the changes that have been made. Cancelling DISABLEEENCYPTION may leave the database in one of 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
- All encryption policies are removed
- The encryption feature is fully disabled

Check your database .lg file or the status messages displayed on screen to determine the state of the database.
Appendixes

Appendix A, Audit Data Tables
Appendix B, Preconfigured Audit Policies
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 in the following section:

- 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.

Figure A–1 illustrates the auditing metadata.

---

Figure A–1: 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 database’s 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.

Table A–1 lists and describes all of the audit tables.

**Table A–1: OpenEdge audit tables**

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_aud-audit-data</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table A–1: OpenEdge audit tables

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_aud-audit-data-value</td>
<td>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 database’s 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.</td>
</tr>
<tr>
<td>_aud-audit-policy</td>
<td>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.</td>
</tr>
</tbody>
</table>
Audit data table schema

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.

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.

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
</table>
| _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. 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. |
### Table A–1: OpenEdge audit tables

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_aud-event-policy</td>
<td>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.</td>
</tr>
<tr>
<td>_aud-field-policy</td>
<td>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.</td>
</tr>
<tr>
<td>_aud-file-policy</td>
<td>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.</td>
</tr>
<tr>
<td>_client-session</td>
<td>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 table’s 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.</td>
</tr>
</tbody>
</table>
Table A–1: OpenEdge audit tables

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
</table>
| `_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. |
| `_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. |
| `_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. |
Table A–2 through Table A–12 describe each of the audit tables individually.

**Table A–2: _aud-audit-data table**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Audit-data-guid</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
<tr>
<td>_Database-connection-id</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
<tr>
<td>_Client-session-uuid</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
<tr>
<td>_User-id</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(35)</td>
</tr>
</tbody>
</table>
### _aud-audit-data table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Audit-date-time</td>
<td>The date and time that the audit event occurred stored as date time timezone, which results in the time recorded to the millisecond.</td>
<td>DATE</td>
<td>99/99/9999</td>
</tr>
<tr>
<td>_Audit-event-group</td>
<td>A GUID pointing at the audit data record containing the audit event group event's 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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
</tbody>
</table>
This facilitates storing extra information about the audit event group without duplicating all the details onto every audit data record.

**Notes:**
- 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.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
</table>
| \_Audit-event-group (continued) | This facilitates storing extra information about the audit event group without duplicating all the details onto every audit data record. **Notes:**
- 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. | CHARACTER | X(28) |
| \_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) |
### Table A–2: _aud-audit-data table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Transaction-id</td>
<td>The database transaction ID for database events.</td>
<td>INTEGER</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>For nondatabase events, this could be an application-supplied transaction sequence, such as a task ID.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> 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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_Transaction-sequence</td>
<td>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.</td>
<td>INTEGER</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_Event-id</td>
<td>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.</td>
<td>INTEGER</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_Event-context</td>
<td>The context within which the event occurred.</td>
<td>CHARACTER</td>
<td>X(3000)</td>
</tr>
<tr>
<td></td>
<td>For application events, this could be a particular procedure or function or an action a user was authorized to access.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>For example:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>database.table + CHR(6) + field1value + CHR(7) + field2value + CHR(7) + field3value</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.

Notes:

- 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 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.

See Table A–5 for more details.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Event-context (continued)</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(3000)</td>
</tr>
<tr>
<td>_Application-context-id</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
</tbody>
</table>
### Table A–2: _aud-audit-data table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Event-detail</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(3000)</td>
</tr>
<tr>
<td>_Audit-custom-detail</td>
<td>A custom field provided to support additional audit data required by an application.</td>
<td>CHARACTER</td>
<td>X(3000)</td>
</tr>
</tbody>
</table>
| _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 policy’s 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 A–3: _aud-audit-data-value table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Audit-data-guid</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
<tr>
<td>_Field-name</td>
<td>The actual name of the field the audit value is for, such as customerName, for example.</td>
<td>CHARACTER</td>
<td>X(32)</td>
</tr>
<tr>
<td>_Continuation-sequence</td>
<td>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.</td>
<td>INTEGER</td>
<td>-&gt;9</td>
</tr>
</tbody>
</table>
Audit data table schema

Table A–3:  _aud-audit-data-value table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Data-type-code</td>
<td>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.</td>
<td>INTEGER</td>
<td>-&gt;9</td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sample data type codes are:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 1 — Character</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 2 — Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 3 — Logical</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 4 — Integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 5 — Decimal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 7 — Recid</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 8 — Raw</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 11 — Memptr</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 13 — Rowid</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 18 — Blob</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 19 — Clob</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 34 — Datetime</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 39 — Longchar</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 40 — Datetime-TZ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_Old-string-value</td>
<td>If the field can be represented as a string value, then this field contains the old string value of the modified field.</td>
<td>CHARACTER</td>
<td>X(3000)</td>
</tr>
</tbody>
</table>
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.

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.

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.
<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Audit-policy-guid</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
<tr>
<td>_Audit-policy-name</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(35)</td>
</tr>
<tr>
<td>_Audit-policy-description</td>
<td>A free text description used to describe the purpose or use of the audit policy.</td>
<td>CHARACTER</td>
<td>X(70)</td>
</tr>
</tbody>
</table>
| _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    |
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.

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.
### Table A–5: _aud-event table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Event-id</td>
<td>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.</td>
<td>INTEGER</td>
<td>-&gt;&gt;&gt;&gt;&gt;9</td>
</tr>
</tbody>
</table>
| _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  
  - 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 | CHARACTER | X(15)   |
| _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 A–6: _aud-event-policy table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Audit-policy-guid</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
<tr>
<td>_Event-id</td>
<td>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.</td>
<td>INTEGER</td>
<td>-&gt;&gt;&gt;&gt;&gt;9</td>
</tr>
</tbody>
</table>
| _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 A–7: _aud-field-policy table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Audit-policy-guid</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
<tr>
<td>_File-Name</td>
<td>This is the actual table name of the database table that the audit policy settings are for.</td>
<td>CHARACTER</td>
<td>X(32)</td>
</tr>
<tr>
<td>_Owner</td>
<td>The schema owner for the table, such as PUB, for example. SQL allows for the same table in different schemas.</td>
<td>CHARACTER</td>
<td>X(32)</td>
</tr>
<tr>
<td>_Field-Name</td>
<td>This is the actual field name of the database field that the audit policy settings are for.</td>
<td>CHARACTER</td>
<td>X(32)</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_Audit-create-level</code></td>
<td>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 <code>_Audit-create-level</code> in the <code>_aud-file-policy</code> table is set to 0), then this field is ignored, as shown:</td>
<td>INTEGER</td>
<td>-&gt;9</td>
</tr>
</tbody>
</table>
This field defines the level of auditing active for this specific field during an update event. If update auditing is turned off at the table level (Audit-update-level in the _aud-field-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-field-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.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Audit-update-level</td>
<td>This field defines the level of auditing active for this specific field during an update event. If update auditing is turned off at the table level (_Audit-update-level in the _aud-field-policy table is set to 0), then this field is ignored, as shown:</td>
<td>INTEGER</td>
<td>-&gt;9</td>
</tr>
<tr>
<td>Field name</td>
<td>Description</td>
<td>Data type</td>
<td>Format</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------</td>
<td>--------</td>
</tr>
<tr>
<td>_Audit-update-level</td>
<td>• 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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(continued)</td>
<td></td>
<td>INTEGER</td>
<td>-&gt;9</td>
</tr>
<tr>
<td></td>
<td>• 12 — Records both record-change events and the new value of only the changed fields in default (streaming) mode, if possible.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 13 — Records both record-change events and the new and old values of only the changed fields in the default (streamed) mode, if possible.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_Audit-delete-level</td>
<td>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:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• –1 — Turns off delete auditing for this field. Does not record any record-change events or field values.</td>
<td>INTEGER</td>
<td>-&gt;9</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table A–7: `_aud-field-policy table` (5 of 5)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
</table>
| `_Audit-delete-level` (continued) | • 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>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Audit-policy-guid</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
<tr>
<td>_File-Name</td>
<td>The actual name of the database table that the audit policy settings are for.</td>
<td>CHARACTER</td>
<td>X(32)</td>
</tr>
<tr>
<td>_Owner</td>
<td>The schema owner for the table, such as PUB, for example. SQL allows for the same table in different schemas.</td>
<td>CHARACTER</td>
<td>X(32)</td>
</tr>
<tr>
<td>_Audit-create-level</td>
<td>The default level of auditing active for the table for create events. The level can have one of the following values:</td>
<td>INTEGER</td>
<td>-&gt;9</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 2 — Standard Audit Data. Same as 1 but additionally stores the initial values in the audit data.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 12 — Records both record-change events and field values in default (streamed) mode, if possible.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Audit data table schema

Table A–8:  _aud-file-policy table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
</table>
| _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:  
  • 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. | INTEGER   | ->9     |
### Table A–8:  _aud-file-policy table  
(3 of 6)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
</table>
| _Audit-update-level (continued) | • **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. | INTEGER   | ->9     |
Audit data table schema

**Table A–8: _aud-file-policy table**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Audit-delete-level</td>
<td>Defines the default level of auditing active for the table for delete events. The level can have one of the following values:</td>
<td>INTEGER</td>
<td>-&gt;9</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 12 — Records both record-change events and field values in default (streamed) mode, if possible.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Field name: `_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.

**Note:** It is invalid to have multiple active audit policies with the same table defined with conflicting event IDs.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>9</td>
</tr>
</tbody>
</table>

### Field name: `_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.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>9</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Delete-event-id</td>
<td>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.</td>
<td>INTEGER</td>
<td>9</td>
</tr>
</tbody>
</table>

### Table A–9: _client-session table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Client-session-uuid</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
<tr>
<td>_Client-name</td>
<td>The name of the type of client session; for example:</td>
<td>CHARACTER</td>
<td>X(35)</td>
</tr>
<tr>
<td></td>
<td>• AS — Appserver</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• WS — WebSpeed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ABL — ABL Client</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SQL — SQL Client</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DBA — Database Admin Tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_User-id</td>
<td>The ID of the authenticated user.</td>
<td>CHARACTER</td>
<td>X(35)</td>
</tr>
<tr>
<td>_Authentication-date-time</td>
<td>The authentication date, time, and time zone.</td>
<td>DATE</td>
<td>99/99/9999</td>
</tr>
</tbody>
</table>
### _client-session table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Server-uuid</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
<tr>
<td>_Authentication-domain-type</td>
<td>The type of authentication plug-in used, such as LDAP, for example.</td>
<td>CHARACTER</td>
<td>X(35)</td>
</tr>
<tr>
<td>_Authentication-domain-name</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(70)</td>
</tr>
<tr>
<td>_Db-guid</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
<tr>
<td>_Session-custom-detail</td>
<td>User-defined session information, such as terminal ID, provided in free text.</td>
<td>CHARACTER</td>
<td>X(3000)</td>
</tr>
</tbody>
</table>
| _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    |
Audit data table schema

Table A–9: _client-session table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Data-seal</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
</tbody>
</table>

Table A–10: _db table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Db-guid</td>
<td>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. <strong>Note:</strong> 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.</td>
<td>CHARACTER</td>
<td>X(28)</td>
</tr>
</tbody>
</table>
Table A–11: _db-detail table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
</table>
| _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)  |
| _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)  |
Audit data table schema

Table A–11: _db-detail table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Db-mac-key</td>
<td>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. <strong>Note:</strong> The MAC key must be stored as an encrypted raw value for security purposes, as it has characteristics similar to a password.</td>
<td>RAW</td>
<td></td>
</tr>
<tr>
<td>_Db-custom-detail</td>
<td>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. <strong>Note:</strong> 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.</td>
<td>CHARACTER</td>
<td>X(3000)</td>
</tr>
</tbody>
</table>
Table A–12:  _db-option table

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
<th>Data type</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_Db-recid</td>
<td>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.</td>
<td>INTEGER</td>
<td>-&gt;&gt;&gt;&gt;&gt;&gt;9</td>
</tr>
<tr>
<td>_Db-option-type</td>
<td>The type of option (for example, security, auditing, or context) used to group related options together. OpenEdge reserves the first 32000 option types.</td>
<td>INTEGER</td>
<td>-&gt;&gt;&gt;&gt;&gt;&gt;9</td>
</tr>
<tr>
<td>_Db-option-code</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(35)</td>
</tr>
<tr>
<td>_Db-option-description</td>
<td>A free-text description of the database option.</td>
<td>CHARACTER</td>
<td>X(500)</td>
</tr>
<tr>
<td>_Db-option-value</td>
<td>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.</td>
<td>CHARACTER</td>
<td>X(70)</td>
</tr>
</tbody>
</table>
Preconfigured Audit Policies

OpenEdge includes several preconfigured audit policies that you can use as is or modify for your specific auditing needs. This appendix describes each of the policies in the following section:

- 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 Table B–1. For more information, see the Audit Policy Maintenance Help.

Table B–1: Predefined OpenEdge database audit policy events

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Event name</th>
<th>Event type</th>
<th>Event description</th>
<th>Policy name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>_sys.user.create</td>
<td>Security</td>
<td>User account created</td>
<td>PSC-UserAccounts</td>
</tr>
<tr>
<td>101</td>
<td>_sys.user.update</td>
<td>Security</td>
<td>User account updated</td>
<td>PSC-UserAccounts</td>
</tr>
<tr>
<td>102</td>
<td>_sys.user.delete</td>
<td>Security</td>
<td>User account deleted</td>
<td>PSC-UserAccounts</td>
</tr>
<tr>
<td>210</td>
<td>_sql.dba.create</td>
<td>Security</td>
<td>SQL DBA created</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>211</td>
<td>_sql.dba.update</td>
<td>Security</td>
<td>SQL DBA updated</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>212</td>
<td>_sql.dba.delete</td>
<td>Security</td>
<td>SQL DBA deleted</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>300</td>
<td>_sys.audit.policy.create</td>
<td>Audit</td>
<td>Audit policy record created</td>
<td>PSC-Audit</td>
</tr>
<tr>
<td>301</td>
<td>_sys.audit.policy.update</td>
<td>Audit</td>
<td>Audit policy record updated</td>
<td>PSC-Audit</td>
</tr>
<tr>
<td>302</td>
<td>_sys.audit.policy.delete</td>
<td>Audit</td>
<td>Audit policy record deleted</td>
<td>PSC-Audit</td>
</tr>
<tr>
<td>400</td>
<td>_sql.tbl.priv.create</td>
<td>Security</td>
<td>SQL table privilege created</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>401</td>
<td>_sql.tbl.priv.update</td>
<td>Security</td>
<td>SQL table privilege updated</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>402</td>
<td>_sql.tbl.priv.delete</td>
<td>Security</td>
<td>SQL table privilege deleted</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>410</td>
<td>_sql.col.priv.create</td>
<td>Security</td>
<td>SQL column privilege created</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>411</td>
<td>_sql.col.priv.update</td>
<td>Security</td>
<td>SQL column privilege updated</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>412</td>
<td>_sql.col.priv.delete</td>
<td>Security</td>
<td>SQL column privilege deleted</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>420</td>
<td>_sql.seq.priv.create</td>
<td>Security</td>
<td>SQL sequence privilege created</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>421</td>
<td>_sql.seq.priv.update</td>
<td>Security</td>
<td>SQL sequence privilege updated</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>422</td>
<td>_sql.seq.priv.delete</td>
<td>Security</td>
<td>SQL sequence privilege deleted</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>500</td>
<td>_sys.auth.sys.create</td>
<td>Security</td>
<td>Authentication system created</td>
<td>PSC-Security</td>
</tr>
</tbody>
</table>
### Table B–1: Predefined OpenEdge database audit policy events

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Event name</th>
<th>Event type</th>
<th>Event description</th>
<th>Policy name</th>
</tr>
</thead>
<tbody>
<tr>
<td>501</td>
<td>_sys.auth.sys.update</td>
<td>Security</td>
<td>Authentication system updated</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>502</td>
<td>_sys.auth.sys.delete</td>
<td>Security</td>
<td>Authentication system deleted</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>505</td>
<td>_sys.auth.domain.create</td>
<td>Security</td>
<td>Authentication domain created</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>506</td>
<td>_sys.auth.domain.update</td>
<td>Security</td>
<td>Authentication domain updated</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>507</td>
<td>_sys.auth.domain.delete</td>
<td>Security</td>
<td>Authentication domain deleted</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>510</td>
<td>_sys.role.create</td>
<td>Security</td>
<td>Role definition created</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>511</td>
<td>_sys.role.update</td>
<td>Security</td>
<td>Role definition updated</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>512</td>
<td>_sys.role.delete</td>
<td>Security</td>
<td>Role definition deleted</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>515</td>
<td>_sys.role.assign.create</td>
<td>Security</td>
<td>Role assignment created</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>516</td>
<td>_sys.role.assign.update</td>
<td>Security</td>
<td>Role assignment updated</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>517</td>
<td>_sys.role.assign.delete</td>
<td>Security</td>
<td>Role assignment deleted</td>
<td>PSC-Security</td>
</tr>
<tr>
<td>5000</td>
<td>_sys.tbl.create</td>
<td>Schema</td>
<td>Table created</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5001</td>
<td>_sys.tbl.trig.create</td>
<td>Schema</td>
<td>Table trigger created</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5002</td>
<td>_sys.fld.create</td>
<td>Schema</td>
<td>Table field created</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5003</td>
<td>_sys.fld.trig.create</td>
<td>Schema</td>
<td>Table field trigger created</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5004</td>
<td>_sys.index.create</td>
<td>Schema</td>
<td>Table index created</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5005</td>
<td>_sys.index.fld.create</td>
<td>Schema</td>
<td>Table index field created</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5010</td>
<td>_sys.tbl.update</td>
<td>Schema</td>
<td>Table updated</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5111</td>
<td>_sys.tbl.trig.update</td>
<td>Schema</td>
<td>Table trigger updated</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5012</td>
<td>_sys.fld.update</td>
<td>Schema</td>
<td>Table field updated</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5013</td>
<td>_sys.fld.trig.update</td>
<td>Schema</td>
<td>Table field trigger updated</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5014</td>
<td>_sys.index.update</td>
<td>Schema</td>
<td>Table index updated</td>
<td>App-Schema</td>
</tr>
<tr>
<td>Event ID</td>
<td>Event name</td>
<td>Event type</td>
<td>Event description</td>
<td>Policy name</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------</td>
<td>------------</td>
<td>--------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>5015</td>
<td>_sys.index.fld.update</td>
<td>Schema</td>
<td>Table index field updated</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5020</td>
<td>_sys.tbl.delete</td>
<td>Schema</td>
<td>Table deleted</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5021</td>
<td>_sys.tbl.trig.delete</td>
<td>Schema</td>
<td>Table trigger deleted</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5022</td>
<td>_sys.fld.delete</td>
<td>Schema</td>
<td>Table field deleted</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5023</td>
<td>_sys.fld.trig.delete</td>
<td>Schema</td>
<td>Table field trigger deleted</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5024</td>
<td>_sys.index.delete</td>
<td>Schema</td>
<td>Table index deleted</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5025</td>
<td>_sys.index.fld.delete</td>
<td>Schema</td>
<td>Table index field deleted</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5030</td>
<td>_sys.seq.create</td>
<td>Schema</td>
<td>Sequence created</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5031</td>
<td>_sys.seq.update</td>
<td>Schema</td>
<td>Sequence updated</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5032</td>
<td>_sys.seq.delete</td>
<td>Schema</td>
<td>Sequence deleted</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5040</td>
<td>_sys.db.prop.create</td>
<td>Schema</td>
<td>Database property created</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5041</td>
<td>_sys.db.prop.update</td>
<td>Schema</td>
<td>Database property updated</td>
<td>App-Schema</td>
</tr>
<tr>
<td>5100</td>
<td>_sys.db.rec.create</td>
<td>Data</td>
<td>Database record created</td>
<td>PSC-App-Tables</td>
</tr>
<tr>
<td>5101</td>
<td>_sys.db.rec.update</td>
<td>Data</td>
<td>Database record updated</td>
<td>PSC-App-Tables</td>
</tr>
<tr>
<td>5102</td>
<td>_sys.db.rec.delete</td>
<td>Data</td>
<td>Database record deleted</td>
<td>PSC-App-Tables</td>
</tr>
<tr>
<td>10000</td>
<td>_sys.db.audit.enable</td>
<td>Audit</td>
<td>Database auditing enabled</td>
<td>PSC-Audit</td>
</tr>
<tr>
<td>10001</td>
<td>_sys.db.audit.disable</td>
<td>Audit</td>
<td>Database auditing disabled</td>
<td>PSC-Audit</td>
</tr>
<tr>
<td>10010</td>
<td>_sys.audit.policy.commit</td>
<td>Audit</td>
<td>Audit policy run-time update</td>
<td>PSC-Audit</td>
</tr>
<tr>
<td>10100</td>
<td>_sys.db.start</td>
<td>Admin</td>
<td>Database started</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10101</td>
<td>_sys.db.stop</td>
<td>Admin</td>
<td>Database stopped</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10200</td>
<td>_sys.backup</td>
<td>Utility</td>
<td>Database backup</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10201</td>
<td>_sys.restore</td>
<td>Utility</td>
<td>Database restore</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10202</td>
<td>_sys.bindump</td>
<td>Utility</td>
<td>Database binary dump</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10203</td>
<td>_sys.binload</td>
<td>Utility</td>
<td>Database binary load</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>Event ID</td>
<td>Event name</td>
<td>Event type</td>
<td>Event description</td>
<td>Policy name</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------</td>
<td>------------</td>
<td>--------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>10204</td>
<td>_sys.copy</td>
<td>Utility</td>
<td>Database copy</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10205</td>
<td>_sys.tbl.move</td>
<td>Utility</td>
<td>Database table move</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10206</td>
<td>_sys.idx.move</td>
<td>Utility</td>
<td>Database index move</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10207</td>
<td>_sys.idx.check</td>
<td>Utility</td>
<td>Database index check</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10208</td>
<td>_sys.idx.rebuild</td>
<td>Utility</td>
<td>Database index rebuild</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10209</td>
<td>_sys.area.truncate</td>
<td>Utility</td>
<td>Database area truncate</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10210</td>
<td>_sys.idx.fix</td>
<td>Utility</td>
<td>Database index fix</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10211</td>
<td>_sys.sql.dump</td>
<td>Utility</td>
<td>Database SQL dump</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10212</td>
<td>_sys.sql.load</td>
<td>Utility</td>
<td>Database SQL load</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10213</td>
<td>_sys.txt.dump</td>
<td>Utility</td>
<td>Database text dump</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10214</td>
<td>_sys.txt.load</td>
<td>Utility</td>
<td>Database text load</td>
<td>PSC-DB-Admin</td>
</tr>
<tr>
<td>10300</td>
<td>_sys.audit.archive</td>
<td>Audit</td>
<td>Audit archive</td>
<td>PSC-Audit</td>
</tr>
<tr>
<td>10301</td>
<td>_sys.audit.data.dump</td>
<td>Audit</td>
<td>Audit data dump</td>
<td>PSC-Audit</td>
</tr>
<tr>
<td>10302</td>
<td>_sys.audit.data.load</td>
<td>Audit</td>
<td>Audit data load</td>
<td>PSC-Audit</td>
</tr>
<tr>
<td>10303</td>
<td>_sys.audit.pol.dump</td>
<td>Audit</td>
<td>Audit policy dump</td>
<td>PSC-Audit</td>
</tr>
<tr>
<td>10304</td>
<td>_sys.audit.pol.load</td>
<td>Audit</td>
<td>Audit policy load</td>
<td>PSC-Audit</td>
</tr>
<tr>
<td>10305</td>
<td>_sys.audit.recoveradmin</td>
<td>Audit</td>
<td>DBA deleted last audit admin</td>
<td>PSC-Audit</td>
</tr>
<tr>
<td>10500</td>
<td>_pvm.setdbuser.success</td>
<td>User</td>
<td>Database user identity set</td>
<td>PSC-User</td>
</tr>
<tr>
<td>10501</td>
<td>_pvm.setdbuser.fail</td>
<td>User</td>
<td>Database user identity failed</td>
<td>PSC-User</td>
</tr>
<tr>
<td>10510</td>
<td>_pvm.user.login.pass</td>
<td>User</td>
<td>User login success</td>
<td>PSC-User</td>
</tr>
<tr>
<td>10511</td>
<td>_pvm.user.logout</td>
<td>User</td>
<td>User logout</td>
<td>PSC-User</td>
</tr>
<tr>
<td>10512</td>
<td>_pvm.user.login.fail</td>
<td>User</td>
<td>User login failed</td>
<td>PSC-User</td>
</tr>
<tr>
<td>10520</td>
<td>_sql.user.login.pass</td>
<td>User</td>
<td>SQL user login success</td>
<td>PSC-User</td>
</tr>
<tr>
<td>10521</td>
<td>_sql.user.logout</td>
<td>User</td>
<td>SQL user logout</td>
<td>PSC-User</td>
</tr>
<tr>
<td>10522</td>
<td>_sql.user.login.fail</td>
<td>User</td>
<td>SQL user login failed</td>
<td>PSC-User</td>
</tr>
<tr>
<td>10600</td>
<td>_pvm.db.connect</td>
<td>User</td>
<td>Database connect</td>
<td>PSC-User</td>
</tr>
<tr>
<td>10601</td>
<td>_pvm.db.disconnect</td>
<td>User</td>
<td>Database disconnect</td>
<td>PSC-User</td>
</tr>
</tbody>
</table>
Table B–1: Predefined OpenEdge database audit policy events  

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Event name</th>
<th>Event type</th>
<th>Event description</th>
<th>Policy name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10610</td>
<td>_sql.db.connect</td>
<td>User</td>
<td>SQL database connect</td>
<td>PSC-User</td>
</tr>
<tr>
<td>10611</td>
<td>_sql.db.disconnect</td>
<td>User</td>
<td>SQL database disconnect</td>
<td>PSC-User</td>
</tr>
<tr>
<td>11000</td>
<td>_sys.db.enc.enable</td>
<td>Encryption</td>
<td>Enable Encrypted Database feature</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11001</td>
<td>_sys.db.enc.disable</td>
<td>Encryption</td>
<td>Disable Encrypted Database feature</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11100</td>
<td>_sys.ks.create</td>
<td>Encryption</td>
<td>Create database key store</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11101</td>
<td>_sys.ks.delete</td>
<td>Encryption</td>
<td>Delete database key store</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11102</td>
<td>_sys.ks.open.pass</td>
<td>Encryption</td>
<td>Successful open database key store</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11103</td>
<td>_sys.ks.rekey</td>
<td>Encryption</td>
<td>Change database key store passphrase</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11104</td>
<td>_sys.ks.setcipher</td>
<td>Encryption</td>
<td>Change database key store master key type</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11105</td>
<td>_sys.ks.setadmin.pwd</td>
<td>Encryption</td>
<td>Change database key store administrator passphrase</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11106</td>
<td>_sys.ks.setuser.pwd</td>
<td>Encryption</td>
<td>Change database key store user passphrase</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11107</td>
<td>_sys.ks.ke.create.pass</td>
<td>Encryption</td>
<td>Successful create of database key store entry</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11108</td>
<td>_sys.ks.ke.update.pass</td>
<td>Encryption</td>
<td>Successful update of database key store entry</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11109</td>
<td>_sys.ks.ke.delete.pass</td>
<td>Encryption</td>
<td>Successful delete of database key store entry</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11110</td>
<td>_sys.ks.ke.read.pass</td>
<td>Encryption</td>
<td>Successful read of database key store entry</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11111</td>
<td>_sys.ks.open.fail</td>
<td>Encryption</td>
<td>Failure in database key store open</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11112</td>
<td>_sys.ks.ke.create.fail</td>
<td>Encryption</td>
<td>Failure in database key store entry create</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11113</td>
<td>_sys.ks.ke.update.fail</td>
<td>Encryption</td>
<td>Failure in database key store entry update</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11114</td>
<td>_sys.ks.ke.delete.fail</td>
<td>Encryption</td>
<td>Failure in database key store entry delete</td>
<td>Database Encryption</td>
</tr>
</tbody>
</table>
Table B–1: Predefined OpenEdge database audit policy events

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Event name</th>
<th>Event type</th>
<th>Event description</th>
<th>Policy name</th>
</tr>
</thead>
<tbody>
<tr>
<td>11200</td>
<td>_sys.as.create.pass</td>
<td>Encryption</td>
<td>Successful database auto-start file create</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11201</td>
<td>_sys.as.delete.pass</td>
<td>Encryption</td>
<td>Successful database auto-start file delete</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11202</td>
<td>_sys.as.open.pass</td>
<td>Encryption</td>
<td>Successful database auto-start file open</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11203</td>
<td>_sys.as.recover.pass</td>
<td>Encryption</td>
<td>Successful database auto-start file recovery</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11204</td>
<td>_sys.as.update.pass</td>
<td>Encryption</td>
<td>Successful database auto-start file update</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11205</td>
<td>_sys.as.open.fail</td>
<td>Encryption</td>
<td>Failure in database auto-start file open</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11206</td>
<td>_sys.as.update.fail</td>
<td>Encryption</td>
<td>Failure in database auto-start file update</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11207</td>
<td>_sys.as.recover.fail</td>
<td>Encryption</td>
<td>Failure in database auto-start file recovery</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11300</td>
<td>_sys.enc.scan</td>
<td>Encryption</td>
<td>Started encryption state scan of database object</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11301</td>
<td>_sys.enc.update</td>
<td>Encryption</td>
<td>Started encryption state migration of database object</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11400</td>
<td>_sys.db.dbpolicy.create</td>
<td>Encryption</td>
<td>Create new version of database security policy</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11401</td>
<td>_sys.db.dbpolicy.update</td>
<td>Encryption</td>
<td>Update database security policy version</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11402</td>
<td>_sys.db.dbpolicy.delete</td>
<td>Encryption</td>
<td>Delete database security policy version</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11500</td>
<td>_sys.db.objpolicy.create</td>
<td>Encryption</td>
<td>Create new version of database object security policy</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11501</td>
<td>_sys.db.objpolicy.update</td>
<td>Encryption</td>
<td>Update database object security policy version</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11502</td>
<td>_sys.db.objpolicy.delete</td>
<td>Encryption</td>
<td>Delete database object security policy version</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11600</td>
<td>_sys.db.pwdpolicy.create</td>
<td>Encryption</td>
<td>Create new database passphrase policy</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>11601</td>
<td>_sys.db.pwdpolicy.update</td>
<td>Encryption</td>
<td>Update database passphrase policy</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>Event ID</td>
<td>Event name</td>
<td>Event type</td>
<td>Event description</td>
<td>Policy name</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------</td>
<td>------------</td>
<td>----------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>11602</td>
<td>_sys.db.pwdpolicy.delete</td>
<td>Encryption</td>
<td>Delete database passphrase policy</td>
<td>Database Encryption</td>
</tr>
<tr>
<td>31998</td>
<td>_sys.app.context</td>
<td>Application</td>
<td>Application context</td>
<td>Application</td>
</tr>
<tr>
<td>31999</td>
<td>_sys.app.eventgroup</td>
<td>Application</td>
<td>Audit event group</td>
<td>Application</td>
</tr>
<tr>
<td>32000 and higher</td>
<td>user-defined</td>
<td>–</td>
<td>User-defined event</td>
<td>–</td>
</tr>
</tbody>
</table>
Index

A

Access code
domain 4–5

AES 3–3

Algorithms
asymmetric-key 3–9
cryptographic
hashing 3–2
message digest 3–10
PBE 3–7
SSL 6–5
symmetric-key
encryption only 3–10
encryption/decryption 3–3
modes 3–4

Application
audit-enabling 11–3
authentication and authorization 2–18
context
auditing 8–4, 10–7, 11–4
defined events
auditing 8–4
deployment security 2–4
development security 2–2
feature authorization 2–8
network security 2–16
portability 2–4
trusted registry 4–6
updating for auditing 11–6
user identity 4–3

Application event inserter privilege 9–4

Application services security 2–9

AppServer
connection security 2–10
firewall compatibility 2–10
program security 2–9

Architecture of OpenEdge SSL 7–2

Archiving audit data 9–9, 9–10, 9–20

Asserting user ID
in synchronization 4–10
singularly 4–10

Assigning a database a unique identifier 9–22

Asymmetric-key cryptography 3–8
algorithms 3–9
keys 3–8
OpenEdge usage 3–9
PKI support 5–5

Audit administrator
primary 9–5
privileges 9–4
secondary 9–5

Audit data archiver privilege 9–4, 9–20

Audit data archiving 9–9, 9–10, 9–20

Audit data reporter privilege 9–4

Audit event groups 10–6, 14–8

Audit policies
exporting 9–14
resolving conflicts 9–13
security 9–12

Audit Policy Maintenance 10–3, 10–4
Audit-event groups
in application auditing 8–4
Auditing
security 2–17
Authentication
ABL support 2–18
application network 2–16
AppServer applications 2–9
defined 1–2
OpenEdge ABL applications 2–2
PKI support 5–2
Progress Dynamics 2–13
Report Builder 2–4
trusted systems and domains 4–4
user
external 4–4, 9–15, 11–4
in OpenEdge 4–3
internal 4–4
WebClient applications 2–4
WebSpeed applications 2–12
Authorities for trust 5–6
Authorization
ABL support 2–18
defined 1–3

B
Basic crypto 3–1
Blank user ID
disallowing 4–11

C
CA 5–6
CA digital certificate 5–7
Cashing SSL sessions 6–5
Certificate
revocation 5–8
stores 5–8
Certificate Authority 5–6
Certificate store
entries 5–8
Certificate store location property 7–13
Certificates, See Digital certificates
Certification Authority (CA) 5–6
Certify 5–6
Ciphers 3–2
See also Algorithms
Client entity 3–8
client-principal object 4–8, 9–18, 9–19
Clients
database
OpenEdge
accessing SSL server X.500 Subject
Name 7–16
configuring and running SSL 7–13
setting SSL properties and parameters
7–14
SSL properties and parameters 7–13
SSL 6–4
Common business service 1–1
Confidentiality
defined 1–4
PKI support 5–2
Context
auditing
database transaction 10–5, 10–6
Continuation records 14–11
Core business service
defined 1–1
Core security services 5–2
Cryptographic
hash functions 3–10
keys 3–2
Cryptography
algorithms 3–2
asymmetric-key 3–8
PKI support 5–5
basic 3–1
data management 3–12
hashing 3–10
keys 3–2
message digests 3–10
PKI support 5–4
OpenEdge ABL 3–6
PKI support
symmetric-key 3–3
PKI support 5–4
Custom
audit archive tool 11–5
audit policy tool 11–5, 11–6
audit reports 12–6
D

Data
  integrity defined 1–4
  integrity in a PKI 5–3
  privacy defined 1–4
  security levels in OpenEdge 2–5

Data management security 2–5

Data privacy
  PKI support 5–2

Database
  audit security 9–10
  auditing events 8–2
  connection identity 4–2
  domain registry 4–6
  transaction context 10–6

Database security 2–6

DataServer security 2–6

Decryption 3–1

Default SSL identity 7–8

default_server alias 7–8

DES 3–3

DES3 3–3

Digital certificates 5–6
  life time 5–7
  life-cycle management 5–8
  revocation 5–8
  SSL 6–4
  standards 5–6
  storage and management 5–8
  X.509 public-key 5–7

Digital signatures 5–7
  in SSL 6–6

Disabling auditing 13–3

Disallowing blank user ID 4–11

Domains
  access code 4–5
  defined 4–5
  trusted by OpenEdge 4–5
  trusted registry 4–6, 9–18

DSA 3–9

E

Enabling a database for auditing 10–2

Encryption 3–1, 15–1

Enforce audit insert privilege option 9–15

Entities
  client and server 3–8

Expired digital certificates 5–7
  managing 5–8

Exporting audit policy configuration 9–14

F

Firewall compatibility
  AppServer 2–10
  SonicMQ Adapter 2–11
  WebSpeed 2–12

G

Granting audit privileges
  in SQL 9–7
  in the 4GL 9–6

H

Hash functions 3–10

Hashing algorithms 3–2

I

Identifying fields 14–6

Identity management in OpenEdge 4–1

Integration services security 2–11

Integrity
  defined 1–4
  PKI support 5–3

Internal events
  auditing 8–3

K

Key alias
  name property 7–11
  password property 7–11

Key pair
  PKI authentication 5–2
  public-key cryptography 3–8

Key store
  entries 7–7
  defined 5–8
  OpenEdge
    default alias 7–8
Key store rebind 16–9
Key store reconstruct 16–10
Key stores
 defined 5–8
Keys
 asymmetric 3–8
cryptographic 3–2
private 3–8
public 3–8
secret 3–3
SSL 6–4
symmetric 3–3

L
Lifetime
digital certificate 5–7
Login sessions
 controlling in audit data records 9–18

M
Managing cryptographic data 3–12
MD5 3–10
Message Authentication Code (MAC), See
Message digests
Message digests 3–10
 OpenEdge usage 3–10
PKI support 5–4
Modes of operation
 cryptographic 3–4

N
Nonrepudiation
 defined 1–2
 in auditing 9–2
PKI support 5–5

O
Open Client application security 2–3
OpenEdge ABL
 application security 2–2
 run-time procedure authorization 2–8
 source code security 2–4
 Web services security 2–10
OpenEdge auditing
 application deployment 12–2, 12–7
 application events 8–2, 11–2
 application-context 8–4, 11–4, 14–7
 application-defined 8–4
archiving 9–9, 9–20
audit event groups 8–4, 10–6, 14–8
Audit Policy Maintenance 10–3, 10–4
context 10–5
 architecture 10–5
 database 10–6
 copying database when 13–2
data context 14–6
data integrity 13–3
data maintenance 13–2
data records 9–18
data recovery 13–2
data tables
 schema A–1
database backing up and restoring 13–2
database events 8–2
disabling 13–3
enabling application for 11–3
enabling database for 10–2, 11–2
enforce audit insert privilege 9–15
evaluating requirements for
 as developer 8–5
 as end user 8–6
exporting audit policy configuration 9–14
features 8–2
getting started 10–2
identifying fields 14–6
implementation recommendations 8–7
indexes 14–14
deactivating 14–5
modifying 13–2
internal events 8–2
performance 13–4
policy conflict resolution 9–13
pre-10.1 databases or clients 8–8
preconfigured policies 10–3, B–1
privileges 9–3
 application audit event inserter 9–4
 audit administrator 9–4
 audit data archiver 9–4, 9–20
 audit data reporter 9–4
 database administrator 9–3
 granting in 4GL 9–5, 9–6
 granting in SQL 9–5, 9–7
 granting to others 9–5
 revoking 9–7
production site configuration 12–3
query guidelines 14–16
querying data 13–5, 14–2
READ 11–4
reports 12–6, 14–4, 14–5
 audit event details 14–8
 by audit date and time 14–9
 client-session information 14–6
custom 14–17, 14–21
database events 14–12
event details 14–9
field values 14–10
internationalization 14–4
orientation 14–17
prefiltered 14–17
prefiltered, list of 14–18
run-time maintenance 13–5
schema overview 14–2
sealing data 9–12
security
database utilities 9–10
function 1–3
policy 9–12
separation of duty 9–2
tools 9–10
utilities 9–10
storage management 8–9, 11–4, 13–2, 13–3
upgrading existing application for 12–8
user ID in records 9–18
user identity 4–3
using application user ID 9–15, 9–19
OpenEdge identity management 4–1
OpenEdge Key Store 16–4
OpenEdge security
application
deployment 2–4
development 2–2
feature authorization 2–8
network 2–16
portability 2–4
application services 2–9
AppServer
connections 2–10
firewall compatibility 2–10
program 2–9
asserting user ID 4–10
asymmetric-key cryptography 3–9
auditing 2–17, 9–1
auditing archive 9–9
authentication 4–3
external 4–4
internal 4–4
authentication and authorization 2–18
CAN permissions 4–5
client-principal object 4–8, 9–18
data management 2–5
database 2–6
DataServer 2–6
external authentication 4–4
identity management 4–1
integration services 2–11
levels for data management 2–5
Open Client applications 2–3
OpenEdge ABL
applications 2–2
run-time procedure authorization 2–8
source code 2–4
Web services 2–10
overview 2–1
Password-based encryption (PBE) 3–7
Progress Dynamics 2–13
application options 2–14
basic settings 2–13
repository structures 2–15
security managers 2–15
Query/Results reporting 2–8
Report Builder 2–4
See also OpenEdge SSL
SQL data 2–7
symmetric-key cryptography 3–6
WebClient applications 2–4
WebSpeed 2–12
OpenEdge session
user identity 4–2
OpenEdge SSL 7–1
algorithms 6–5
architecture 7–2
non-OpenEdge client and server components 7–4
OpenEdge client and server components 7–3
client session properties 7–13
setting 7–14
clients
configuring 7–13
configuring and running sessions 7–10
features and services 6–3
key and certificate management 6–6
procedure for using 7–6
server session properties 7–11
setting 7–12
servers
configuring 7–11
default identity 7–8
establishing identity 7–7
identity management 7–7
trusted CA identity 7–8
X.500 Subject Name 7–16
session caching 6–5
SSL standards supported 6–2
support limitations 6–3
user interactions 6–7
Password 1–2
Password-based encryption (PBE) 3–7
OpenEdge usage 3–7
PKI support 5–5
PBE See Password-based encryption (PBE)
Permissions 1–3
PKCS#5 3–7
PKI trust model 5–6
PKI, See Public-Key Infrastructure (PKI) 5–1
digital signatures 5–7
key and certificate management 5–8

Preconfigured audit policies 10–3

Primary audit administrator 9–5

Private key 3–8
storage and management 5–8

Privileges
auditing in OpenEdge 9–3
application audit event inserter 9–4
audit administrator 9–4
audit archiver 9–4, 9–20
audit data reporter 9–4
database administrator 9–3
granting 9–6
granting to others 9–5
inheriting 9–3
revoking 9–7
security 9–4
granting in SQL 9–5
granting in the 4GL 9–5

PRNG 5–4

Procedure for using OpenEdge SSL 7–6

Progress Dynamics security 2–13
application options 2–14
basic settings 2–13
repository structures 2–15
security managers 2–15

Properties and parameters
OpenEdge SSL clients 7–13
setting 7–14
OpenEdge SSL server 7–11
setting 7–12

Pseudo Random Number Generation function (PRNG) 5–4

Public key 3–8

Public-key cryptography 3–8
PKI support 5–5
See also Asymmetric-key cryptography

Public-Key Infrastructure (PKI) 5–1
core services supported 5–2
authentication 5–2
confidentiality 5–2
integrity 5–3
cryptography 5–4
asymmetric-key 3–8, 5–5
See also Cryptography
symmetric-key 5–4
password-based encryption (PBE)
PKI support 5–5
support for SSL 6–3

trust models and mechanisms 5–6
client and server interactions 5–7
digital certificates 5–7

Query/Results reporting security 2–8
Querying audit data 14–2

READ auditing
setting up 11–4

Recording authenticated client sessions 4–11

Registries
application 4–6
database domain 4–6
trusted domain 4–6

Report Builder security 2–4

Resolving audit policy conflicts 9–13

Reuse a disconnected SSL session property 7–13

Revoking audit privileges 9–7

Root CA 5–6

Root CA digital certificate 5–7

Root certificate store entries 7–7

RSA 3–9

Sealing the audit data 9–12

Secondary audit administrator 9–5

Secret key 3–3

Secret-key cryptography 3–3
PKI support 5–4

Secure Sockets Layer (SSL) 6–1
features and services 6–3
OpenEdge
client session properties 7–13
server session properties 7–11

OpenEdge limitations 6–3
session components 6–4
algorithms 6–5
caching 6–5
identity 6–4
keys and certificates 6–4
standards supported 6–2
trust support 6–6
user interactions 6–7

Security
auditing 1–3, 9–1
authentication 1–2
authorization 1–3
confidentiality 1–4
integrity 1–4
OpenEdge support 2–1
overview 1–2
trust 1–5

Security policies
OpenEdge ABL 3–6
SSL 6–4

SECURITY-POLICY 4–8

Server digital certificate 5–7

Server entity 3–8

Servers
OpenEdge
configuring and running SSL 7–11
default SSL identity 7–8
managing SSL identity 7–7
setting SSL properties and parameters 7–12
SSL properties and parameters 7–11
trusted CA identity 7–8
SSL 6–4

Session ID
SSL 6–4

Session time out property 7–11

SET-CLIENT( ) method 4–8, 9–19

SET-DB-CLIENT function 4–8, 9–19

SHA-1 3–10

Signing
digital 5–7

SQL
granting auditing privileges 9–7

SQL application
audit-enabling 11–6

SQL data security 2–7

SSL client 6–4

SSL connections enabled property 7–11

SSL server 6–4

SSL sessions 6–4
caching 6–5
configuring and running OpenEdge 7–10

SSL, See Secure Sockets Layer (SSL)

Storage management
OpenEdge audit data 8–9

Strength of cryptography 3–2

Symmetric-key cryptography 3–3
encryption/decryption algorithms 3–3
encryption-only algorithms 3–10
keys 3–3
OpenEdge usage 3–6
PKI support 5–4

Synchronization
in asserting user IDs 4–10

T

Transparent Data Encryption 15–1
Autostart 16–6
definition 15–3
Disabling encryption 17–17
Enabling encryption 15–12
encryptable objects 15–9
Encryption policy rekey 17–10
Key store reconstruct 16–10
Keystore rebind 16–9
Manual start 16–5
OpenEdge Key Store 16–4
OpenEdge SQL support 17–13
passphrase constraints 15–14
Supported ciphers 15–13

Triple DES 3–3

Trust
models and relationships 1–5
PKI support 5–6
SSL support 6–6

Trusted authentication systems and domains 4–4

Trusted CA 5–6

Trusted domain registry 4–6, 9–18

U

Unique identifier (UUID)
assigning to a database 9–22

Upgrading
existing application for auditing 12–8

Use a session cache property 7–11
Use Application User ID for Auditing option 9–15

Use SSL to connect property 7–13

User identity
application 4–3
asserting in OpenEdge 9–2
auditing 4–3
controlling in audit records 9–18
database connection 4–2
managing 4–2
OpenEdge session 4–2
types in OpenEdge 4–2

User interactions
SSL 6–7

User login session context 10–7

User name 1–2

V
Verify the SSL server host name property 7–13

W
Web services security 2–10
WebClient security 2–4
WebSpeed security 2–12

X
X.500 Subject Name 7–16
X.509
certificates 5–7
in SSL 6–6
standard 5–6